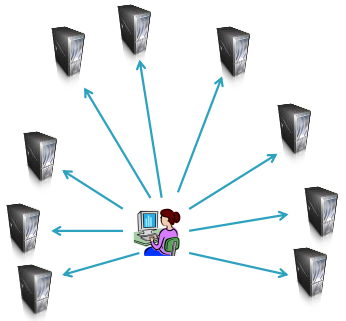
Advanced Web Security

Web Based Single Sign-On and Access Control

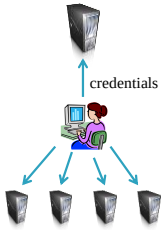
Authenticating to Websites

- Different username and password for each website
- Typically, passwords
  - will be reused
  - will be weak
  - will be written down
- Many websites to attack when looking for passwords
  - Users need to trust them all
    - Will they use salt
    - Will hash function be slow
    - Will they even be hashed
- Stolen hashed passwords
  - Will be easier to recover
  - Can be tested on other websites



Single Sign-On

- Authenticate once – use many times
  - Typical solution – cookies
  - Cookies do not cross domains
- Need solution that can work between domains
  - Many solutions to this problem
    - Main ideas very similar
    - Three parties involved
- We look at SAML, OpenID
  - Several other variants exists



Involved Parties

Description of party	SAML name (Web based SSO profile)	OpenID name
The party that authenticates the user	Identitiy Provider (IdP)	OpenID Provider (OP)
The user of the system	Subject	Subject
The party that the user wants to log in to	Service Provider (SP)	Relying Party (RP)

SAML

- Security Assertion Markup Language
- Based on XML

Uses four main notations

- Assertions – Statement about a subject
  - Authentication statement
  - Attribute statement
  - Authorization decision statement
- Protocols – How assertion should be exchanged
- Bindings – How assertion should be transported
  - HTTP GET, HTTP POST, SOAP, ...
- Profiles – How assertions, protocols and bindings should be used in a particular scenario
  - Web based single sign-on is one profile

SAML Assertions

```
<saml:assertion>
  <saml:Issuer>
  <saml:Subject>
  <saml:Conditions>
  <saml:AuthnStatement>
  <saml:AttributeStatement>
  <saml:AuthzDecisionStatement>
</saml:assertion>
```

String defining the issuer of the assertion

String defining the subject of the assertion

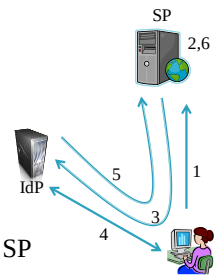
Conditions under which the assertion is valid
- Timestamps (notBefore and notAfter)
- Who should use the assertion

Assertions (one or more)

- Authentication statement
  - The assertion subject was authenticated at a particular time by some particular means
- Attribute Statement
  - The assertion subject is associated with some particular attributes
- Authorization Decision Statement
  - Decision that a subject has been authorized access to some particular resource (or not)

Sign-On

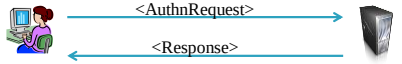
1. Subject attempts to access resource on SP
2. SP determines identity of IdP
3. SP sends <AuthnRequest> to IdP
4. IdP authenticates the subject
5. IdP sends <Response> back to the SP (Authentication Statement)
6. SP verifies the authentication statement



- Steps 3 and 5 can be sent directly to receiver

SAML Authentication Request Protocol

- Authentication request protocol defines how to ask for an assertion



Example of request

```
<AuthnRequest ID="…">
  <Subject>
    ...
  </Subject>
  <Conditions>
    ...
  </Conditions>
  <Issuer>..</Issuer>
</AuthnRequest>
```

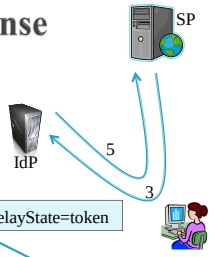
Example of response

```
<Response>
  <Assertion>
    ...
    <Subject>
      ...
    </Subject>
    <AuthnStatement>
      ...
    </AuthnStatement>
  </Assertion>
</Response>
```

- Binding defines how these messages are sent to the receiver

Sending Request and Response

- Binding defines how these are sent
  - **HTTP redirect** – Browser is redirected to destination
    - Only used for requests



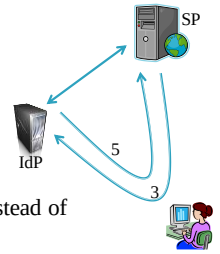
http://IdP-example.com/redirect?SAMLRequest=req&RelayState=token

Base64 encoding of SAML message

  - **HTTP post** – An HTML form is created and posted to destination
    - Example request (response is similar)

```
<form method="post" action="https://IdP-example.com/redirect">
<input type="hidden" name="SAMLRequest" value="req" />
<input type="hidden" name="RelayState" value="token" />
<input type="submit" value="Submit" />
</form>
```


Artifact Binding



- Complete message is not in redirect or in HTML form (not GET or POST)
- Direct requests and responses
- Send artifact (reference) in step 3 or 5 instead of ”actual” message
  - Receiver of artifact makes a connection to sender
  - Request-response is exchanged

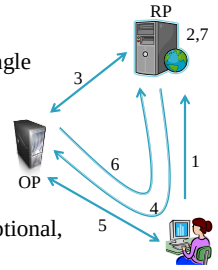
  or
  - Response is sent
- This exchange can use e.g., SOAP if this is something they both support (and prefer)

OpenID

- OpenID 2.0 proposed in 2007
- Lightweight variant of Single Sign-On
- Everything well defined – but still flexible
- Open source

- Suitable when signing in to web pages that provides some personalization

OpenID Steps

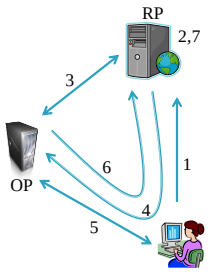


- Very similar to the SAML Web based single sign-on profile

1. Subject attempts to access resource on RP
2. RP performs discovery to locate OP
3. RP and OP establish an association (optional, uses Diffie-Hellman)
4. RP redirects subject to OP (Authentication request)
5. OP authenticates subject/user
6. OP redirects back to RP with signed authentication message (authentication response)
7. RP verifies message and signature

Communication

- Indirect communication
  - Messages are relayed through the user-agent
  - Can be initiated by RP or OP
  - Steps 4 and 6
  - HTTP GET redirect or HTTP POST
- Direct communication
  - Message are communicated directly between RP and OP
  - Can only be initiated by RP using HTTP POST
  - Step 3: RP wants to establish association
  - Step 7: Verification of authentication



Identifiers

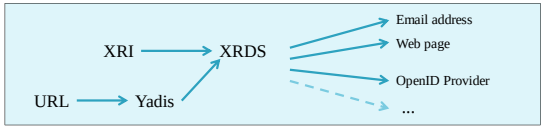
- **OP Identifier** – Identifier for an OpenID provider
  - Defines which OP the end user is using, e.g., http://www.myopenid.com/.
- **Claimed Identifier** – The identifier that the user claims to own
  - RP should use this when saving data about a user (account name at RP)
- **User-Supplied Identifier** – The identifier that the end user presents to the RP
  - Used for discovery. After discovery, the RP will get either an OP Identifier or a Claimed Identifier.
- **OP-Local Identifier** – The identifier that the user has locally with the OP.

XRI and XRDS

- Extensible Resource Identifier
- eXtensible Resource Descriptor Sequence

- XRI is a generalized version of URI
  - Globally unique string with more features than URI
  - Can resolve to anything depending on situation
- Prefix "=" used for people
  - =john.doe is an XRI for person john.doe
- Prefix "@" used for companies
  - @company is XRI for company "company"
- XRI resolves to XRDS document
  - XML document resolving XRI to "anything depending on situation"

Proxy is used for resolving



Discovery, XRDS

- Subject provides an XRI or URL to the RP
  - RP normalizes the User-supplied identifier

- If User-Supplied Identifier is an OP identifier, then XRDS document will give URL to OP

Tells that the provided identifier is an OP identifier

(Part of) XRDS document

```
<Service>
  <Type>http://specs.openid.net/auth/2.0/server</Type>
  <URI>https://exampleOP.com/auth</URI>
</Service>
```

OP Endpoint URL

- User will choose Claimed Identifier upon authenticating with OP

Discovery, XRDS

- If User-Supplied Identifier is a claimed identifier, then XRDS document will give URL to OP and possibly an OP-Local identifier

Tells that the provided identifier is a claimed identifier

(Part of) XRDS document

```
<Service>
  <Type>http://specs.openid.net/auth/2.0/signon</Type>
  <URI>https://exampleOP.com/auth</URI>
  <LocalID>https://john-doe.exampleOP.com/</LocalID>
</Service>
```

OP Endpoint URL

OP-local identifier

- Possible to switch to another OP while keeping claimed identifier

HTML based discovery

- Used when Yadis protocol does not return XRDS document
  - URL is Claimed Identity
  - URL points to HTML document
- Further info in HTML <head>

In HTML <head>

OP Endpoint URL

```
<link rel="openid2.provider openid.server"
      href="https://exampleOP.com/auth"/>
<link rel="openid2.local_id openid.delegate"
      href="https://john-doe.exampleOP.com/"/>
```

OP-local identifier

Authentication Request (made nicer, GET request)

```
GET /accounts/o8/ud?
openid.ns=http://specs.openid.net/auth/2.0&
openid.mode=checkid_setup&
openid.claimed_id=http://specs.openid.net/auth/2.0/identifier_select&
openid.identity=http://specs.openid.net/auth/2.0/identifier_select&
openid.assoc_handle=AMlYA9…&
openid.realm=http://www.someRP.com&
openid.ns.ax=http://openid.net/srv/ax/1.0&
openid.ax.mode=fetch_request&
openid.ax.type.attr0=http://axschema.org/contact/email&
openid.ax.type.attr3=http://axschema.org/namePerson/first&
openid.ax.type.attr6=http://axschema.org/namePerson/last&
…
openid.return_to=http://www.someRP.com/…
```

- OP identifier was entered
- Fixed string
- OP should authenticate subject
- handle to association between OP and RP
- Scope of authentication request
- Attribute info
- Where to return the response

Authentication Response (made nicer, POST data)

```
openid.ns=http://specs.openid.net/auth/2.0&
openid.mode=id_res&
openid.op_endpoint=https://www.google.com/accounts/o8/ud&
openid.response_nonce=2013-12-12T11:10:00Z…A&
openid.return_to=http://www.someRP.com/…&
openid.assoc_handle=AMlYA9…&
openid.signed=op_endpoint,claimed_id,identity,…&
openid.sig=tNxssdjl…ldSTYgQWs=&
openid.identity=https://www…. /id?id=AItO…&
openid.claimed_id=https://www…. /id?id=AItO…&
openid.ns.ext1=http://openid.net/srv/ax/1.0&
openid.ext1.mode=fetch_response&
openid.ext1.type.attr3=http://axschema.org/namePerson/first&
openid.ext1.value.attr3=Martin&
```

- Fixed string
- Fixed string
- OP URL
- Replay protection
- Same as in request
- handle to association between OP and RP
- What was signed
- Signature (Base64)
- OP Local identifier
- Claimed identifier
- Attribute info

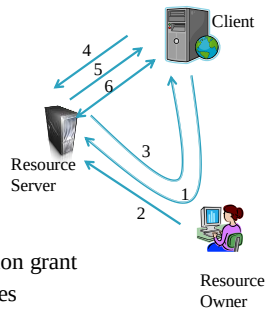
OAuth

- Allow one service to access a user's information on another server
  - Assume that SSL/TLS is used at all times
- **Resource Server** – The server that hosts the resources that a client will get access to.
- **Resource Owner** – An end user that uses the resource server to store some resources, e.g., files or information.
- **Client** – The party that gets (limited) access to resources on the resource server on behalf of the resource owner
- **Authorization Server** – The server that issues access tokens for the resource server to the client
  - Can be the same as the resource server (this will be the case in the examples here)

Steps In Protocol

One way of doing it

1. Client redirects Resource Owner to Resource Server (authorization request)
2. Resource Owner authenticates to Resource Server and gives Client access
3. Resource owner issues authorization grant
4. Client sends grant and authenticates
5. Resource server issues access token
6. Token used to access resources



Obtaining The Grant

- One type of grant is an authorization code
- Authorization request using a HTTP GET redirect

```
GET /oauthreq?response_type=code&client_id=nSv89drLh
    &state=jSasdhfia4y&scope=read,make
    &redirect_uri=https%3A%2F%2Fclient%2Ecom%2Foauth HTTP/1.1
Host: server.com
```

Response_type gives type of grant
Client_id identifies the client
State can be used by client to maintain a state

Scope specifies requested access
Redirect_uri says where to return grant

- Code is returned to Client

```
GET /oauth?code=hdjE75hjGDbsju35h9&state=jSasdhfia4y HTTP/1.1
Host: client.com
```


Requesting Access Token

- Request sent in a HTTP POST
  - Can authenticate using e.g., Basic Access Authentication

```
POST /tokenreq HTTP/1.1
Host: server.com
Authorization: Basic c2VydmVyLmNvbTpwYXNzd29yZA==
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=hdjE75hjGDbsju35h9
&redirect_uri=https%3A%2F%2Fclient%2Ecom%2Foauth
```

grant_type gives type of grant
code is the actual grant
redirect_uri is same as when requesting token

Returning Access Token

- Returned to client in JSON format

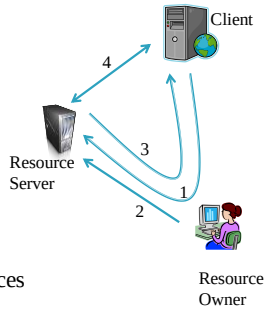
```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"Gfr53SSwfwUnb9kGd4XaeCBV",
  "token_type":"example",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
  "scope":"read"
}
```

- Refresh_token can be used to obtain a new token once the current has expired

Other Grants – Implicit Grant

- **Implicit grant** – Token is returned immediately without making an explicit grant first
  - Makes sense if Client is e.g., a javascript



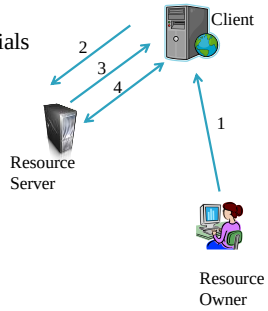
1. Client redirects Resource Owner to Resource Server
2. Resource Owner authenticates
3. Access token is returned
4. Access token used to access resources

- Client does not authenticate

Other Grants - Resource Owner Password Credentials

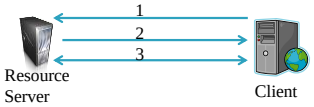
- Resource owner gives the username and password to the client
  - Should only be used if Resource Owner fully trusts the Client

1. Resource owner sends credentials to Client
2. Client authenticates and sends owner's credentials
3. Resource Server issues an access token
4. Client can access resources



Other Grants - Client Credentials

- Client obtains token without going through the resource owner first



1. Client authenticates and requests access token
2. Access token is issued
3. Client can access resources using the token

OpenID connect

- OAuth 2.0 is designed for authorization
- OpenID is designed for authentication

- OpenID connect uses Oauth 2.0 in order to provide authentication
  - Launched 2014

- Separate authorization server and resource server
  - Authorization server is IdP
  - Resource server is Service Provider