# Shibboleth SP Hands-on

Shilen Patel - shilen@duke.edu
Rob Carter - rob@duke.edu
Gonzalo Guzman - gonz@mcnc.org

## Credits and Acknowledgements

- These slides were created by Lukas Hämmerle and Chad La Joie from SWITCHaai.
  - They created their slides using a set created by Scott Cantor from Internet 2.
- Adapted by Shilen Patel for this presentation.
- If you see this on a slide, hands-on work is required

## Tips and Tricks for Hands-on Part

- Restart the Shibboleth daemon shibd after every change
  - shibd automatically reloads config but only restarts “reveal” errors
  - Alternatively, you could just look at the log file for errors
- Also restart your browser cookies after changes
  - Shouldn’t be necessary either in most cases but is safer that way
- Use SSH to connect to VM

  ```
  $ ssh root@spXXX.example.org
  ```

  Open two ssh connections (terminals) to your VM

  Then use `$ tail -f /var/log/shibd.log` on one terminal

## Debugging SP problems

- Make sure the edited XML config file is valid XML

```
$ /usr/sbin/shibd -tc /etc/shibboleth/shibboleth2.xml
$ xmlwf /etc/shibboleth/shibboleth2.xml
```

- Stop Shibboleth daemon

```
$ /etc/init.d/shibd stop
```

- Increase log verbosity of shibd
  - Set log level to DEBUG in `/etc/shibboleth/shibd.logger`
- Have a look at log file

```
$ tail -f /var/log/shibboleth/shibd.log
```

  - Check for any ERROR or CRIT messages
- Start Shibboleth daemon again

```
$ /etc/init.d/shibd start
```

- If you fixed an error, also restart Apache

```
$ /etc/init.d/httpd restart
```

## Setup preparation

1. Run the following script:
   - /opt/installfest/prepare_sp_hands_on.sh

2. Add the following to the bottom of /etc/httpd/conf/httpd.conf

```
<Directory "/var/www/html">
    AllowOverride AuthConfig
    Order allow,deny
    Allow from all
</Directory>
```

3. Modify your IdP configuration to release the following attributes to all SPs:
   - givenName
   - commonName (cn)
   - email (mail)
   - eduPersonEntitlement
   - isMemberOf

   You can do this by replacing your attribute-resolver.xml and attribute-filter.xml file with the following:

   http://www.duke.edu/~shilen/shibtraining/attribute-filter.xml
   http://www.duke.edu/~shilen/shibtraining/attribute-resolver.xml

## Main Goals of Hands-On Session

- Install a Shibboleth Service Provider 2.0
- Know how and where to configure things
- Learn how to protect static web pages
- Understand how attributes can be used in web applications
- How to add a separate Shibboleth application

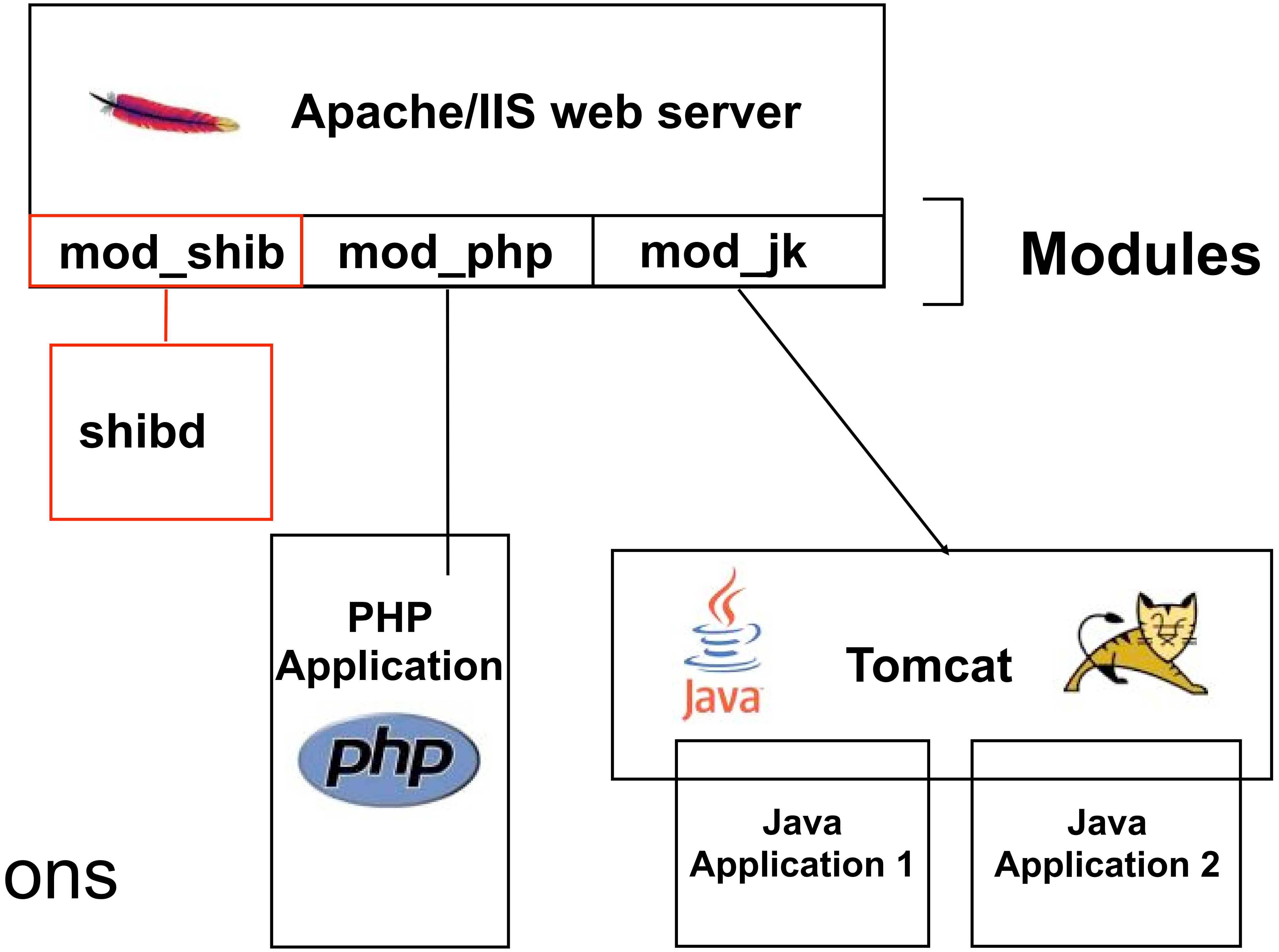
## SP Overview and Installation

**Goals:**

1. Terminology and SP Overview
2. Installation and Directory Structure
3. Generating Key and Certificate
4. Quick Sanity Check
5. Picking an entityID

## SP daemon & mod_shib for Apache

- Runs on: Linux, Solaris, Windows, Mac OS X, FreeBSD, …
- Protects web applications
- Gets attributes from shibd
- Can authorize users with
  - Apache directives
  - Shibboleth XML Access rules
- Provides attributes to applications



## Terminology

- **Service Provider (SP)**
  Consumes SAML assertions, protects web applications
- **Identity Provider (IdP)**
  Asserts digital identities using SAML
- **Discovery Service/WAYF (DS/WAYF)**
  Let’s user choose home organization
- **shibd** (Shibboleth daemon)
  SP service/daemon for maintaining state
- **Session**
  Security context and cached data for a logged-in user
- **SessionInitiator**
  Part of SP that generates SSO requests

## Service Provider Installation

- RPM-based:

```
$ rpm -ivh /opt/installfest/distro/RPMS/*.rpm
```

- Done by RPM after installation:
  - `apache22.conf` copied to `/etc/httpd/conf.d/shib.conf`
  - `shibd` init script copied to `/etc/init.d/shibd`
- Easy-to-install packages also available for other platforms
- Manual compilation not very difficult either
  - But more difficult to maintain efficiently

## Service Provider Binaries

- `/usr/sbin/shibd`

  Shibboleth daemon

- `/usr/bin/resolvertest`

  Resolves attributes from local DB

- `/usr/bin/mdquery`

  Queries metadata of a specified entity/role

- `/usr/lib/shibboleth/*.so`

  Apache/etc. modules, SP extensions

## Sanity Checks

- Start processes:

```
$ /etc/init.d/shibd start
$ /etc/init.d/httpd restart
```

- Check status locally from VM shell (should return some XML):

```
$ curl -k \
  https://spXXX.example.org/Shibboleth.sso/Status \
  --interface lo
```

- Access session handler from your browser:

  https://spXXX.example.org/Shibboleth.sso/Session

  You should get “A valid Session was not found” error

- See how a Shibboleth error looks like (you get an exception):

  https://spXXX.example.org/Shibboleth.sso/Foo

## Important directories

- `/etc/shibboleth/`
  - Master and supporting configuration files
  - Locally maintained metadata files
  - HTML templates (customize them to adapt look&feel to your application)
  - Logging configuration files (*.logger)
  - Credentials (certificates and private keys)
- `/var/run/shibboleth/`
  - UNIX socket
  - remote metadata backups
- `/var/log/shibboleth/`
  - shibd.log and transaction.log files
- `/var/log/httpd/`
  - `native.log` (after permission change)

## Key/Certificate Generation

- Script: `/etc/shibboleth/keygen.sh`
  - May first need: `chmod u+x /etc/shibboleth/keygen.sh`
- Runs automatically during installation on most platforms
- For this event, copy over a pre-generated set for your SP:

```
$ cp /opt/installfest/sps/spXXX/sp.key \
  /etc/shibboleth/sp-key.pem

$ cp /opt/installfest/sps/spXXX/sp.crt \
  /etc/shibboleth/sp-cert.pem
```

If asked, answer yes to overwrite existing files

## Bootstrapping the SP

**Goals:**

1. Make SP communicate with a single test IdP
2. Enable debugging of session attributes

**Note:** Some of the following steps won’t be commented in great detail because they are required only for bootstrapping and will be described later on.

## Picking an entityID for your SP

- Use service FQDN of your application:
  - `https://spXXX.example.org/shibboleth`
- Why use service name?
  - Names should be
    - Unique
    - Locally scoped
    - Logical representative
    - Unchanging
- Where is entityID used?
  - In transmitted messages, local configuration, metadata
  - IdP log files, configuration, filtering policies

## Bootstrapping the SP

- Relax some requirements, set your entityID and default IdP entityID

```
$ vim /etc/shibboleth/shibboleth2.xml


Line 78:
<ApplicationDefaults id="default" policyId="default"
    entityID="https://spXXX.example.org/shibboleth"
    homeURL="https://spXXX.example.org/index.html" >

Line 106:
<SessionInitiator type="Chaining" Location="/Login"
    isDefault="true" id="Intranet" relayState="cookie"
    entityID="https://idpXXX.example.org/idp/shibboleth">

Line 184:
<Handler type="Session" Location="/Session"
    showAttributeValues="true"/>
```

## Bootstrapping the SP (cont.)

- Get the IdP metadata remotely:

```
$ vim /etc/shibboleth/shibboleth2.xml

Line 211:
<MetadataProvider type="Chaining">
   <MetadataProvider type="XML" uri="https://idpXXX.example.org/idpXXX-
       metadata.xml" backingFilePath="/etc/shibboleth/idpXXX-metadata.xml"
       reloadInterval="3600">
   <!-- <SignatureMetadataFilter …  /> -->
   </MetadataProvider>
```

  Comment out the `<SignatureMetadataFilter>` element!

- Normally: Provide your SP's metadata to federation/IdPs
  - Metadata self-generated by your Service Provider
  - `https://spXXX.example.org/Shibboleth.sso/Metadata`

## Quick Test

- Make sure configuration works ( should return “is loadable”):

```
$ shibd -tc /etc/shibboleth/shibboleth2.xml
```

  Service Provider reloads configuration automatically when touched

- Try it with a browser:

  https://spXXX.example.org/secure/

  `/secure/` is protected by Shibboleth “by default”. See bottom of file `/etc/httpd/conf.d/shib.conf`
  Therefore, you should be forced to authenticate. Login at Test IdP with student1/password and you should get access to this directory. Though you will get a 404 error because there’s no content there.

- Then call the Shibboleth session handler to see your attributes:

  https://spXXX.example.org/Shibboleth.sso/Session
  You should see various attributes like affiliation, entitlement, eppn, etc.

## How to use a Discovery Service (WAYF)

- Change the default SessionInitiator:

```
$ vim /etc/shibboleth/shibboleth2.xml

Line 105:
<SessionInitiator type="Chaining" Location="/Login"
    isDefault="false" id="Intranet" relayState="cookie"

Line 119:
<SessionInitiator type="Chaining" Location="/DS" id="DS"
    relayState="cookie" isDefault="true" acsByIndex="false">
    […]
   <SessionInitiator type="SAMLDS" URL="https://ds.example.org/DS/
    WAYF"/>
</SessionInitiator>
```

- Now if you access the SP, instead of being redirected to the IdP, you will be redirected to the Discovery Service.

## Basic configuration

**Goals:**

1. Understand purpose and structure of different configuration files
2. Enable Debug logging
3. Configure metadata and add signature verification

## Configuration Files in /etc/shibboleth

- **`shibboleth2.xml` – main configuration file**
- `apache*.config` – Apache module loading
- `attribute-map.xml` – attribute handling
- `attribute-policy.xml` – attribute filtering
- `*.logger` – logging levels
- `*Error.html` – error templates
- `localLogout.html` – SP-only logout template
- `globalLogout.html` – single logout template

**Recommendation:**
Adapting *.html files to match the look & feel of the application that is protected by Shibboleth improves user experience.

## shibboleth2.xml Structure

Outer elements of the shibboleth.xml configuration file

- <OutOfProcess> / <InProcess>

- <UnixListener> / <TCPListener>

- <StorageService>
- <SessionCache>
- <ReplayCache>
- <ArtifactMap>

- **<RequestMapper>** Needed for session initiation and access control
- 
- **<ApplicationDefaults>** Contains the most important settings of your SP
  - **More on the next slide…**

- <SecurityPolicies>

## ApplicationDefaults Structure

You are most likely to change something in here:

- <ApplicationDefaults>
  - **<Sessions>** Defines handlers and how sessions are initiated and managed
  - **<Errors>** Used to display error messages. Provide here logo, email and CSS
  - <RelyingParty> (*) To modify settings for certain IdPs/federations
  - **<MetadataProvider>** Defines the metadata to be used by the SP
  - <TrustEngine> Which mechanisms to use for signatures validation
  - <AttributeExtractor> Attribute map file to use
  - <AttributeResolver> By default, query the IdP for attributes if none included during SSO.
  - <AttributeFilter> Attribute filter file to use
  - **<CredentialResolver>** Defines certificate and private key to be use
  - <ApplicationOverride> (*) Can override any of the above for certain applications

* = optional

## Logging

- Your number one friend in case of problems
- Logging controlled independently between Apache and shibd (or globally to a syslog)
- `*.logger` files contain predefined settings for output locations and a default logging level (INFO) along with useful categories to raise to DEBUG
- Log time is in UTC (Coordinated Universal Time)

## Logging: Tracing Messages

- Raise categories:

```
$ vim /etc/shibboleth/shibd.logger
```

Line 2:

```
log4j.rootCategory=DEBUG, shibd_log
```

Line 14:

```
# tracing of SAML messages and security policies
log4j.category.OpenSAML.MessageDecoder=DEBUG
log4j.category.OpenSAML.MessageEncoder=DEBUG
log4j.category.OpenSAML.SecurityPolicyRule=DEBUG
```

- To make shibd reload `*.logger` changes:

```
$ touch /etc/shibboleth/shibboleth2.xml
$ tail -f /var/log/shibboleth/shibd.log
```

(reloads configuration)

- Logout (close browser), access `/secure/` and have a look at the log file:

  `https://spXXX.example.org/secure`

  You should see the encrypted XML assertion received by your SP.

## SP Metadata Features

- Metadata describes the other components (mostly IdPs) that the Service Provider may communicate with

- **Four primary methods built-in:**
  - Local metadata file (you manage it)
  - Get metadata file from URL (periodic refresh, backed up for you)
  - Dynamic resolution of entityID (=URL)

- Security comes from metadata filtering, either by you or the SP:
  - Signature verification
  - White and blacklists

## Signature Verification

- The IdP's metadata is signed. Up until now, you loaded it without checking, which is not secure and not recommended!
- Here’s how you can verify the signature.

```
<MetadataProvider type="Chaining">
  <MetadataProvider type="XML" uri="https://idpXXX.example.org/
    idpXXX-metadata.xml" backingFilePath=”/etc/shibboleth/idpXXX-
    metadata.xml" reloadInterval="3600">
     <SignatureMetadataFilter certificate="/path/to/idp/cert"/>
  </MetadataProvider>
</MetadataProvider>
```

## Signature Verification continued

- If you add the wrong certificate, you will get the following error....

```
2008-07-17 11:21:12 WARN OpenSAML.MetadataFilter.Signature [3]:
filtering out group at root of instance after failed signature check:
```

```
2008-07-07 11:21:12 ERROR OpenSAML.Metadata.Chaining [3]: failure
initializing MetadataProvider: SignatureMetadataFilter unable to
verify signature at root of metadata instance.
```

- Metadata could not be loaded because it was signed with a different key...

## Attribute Handling

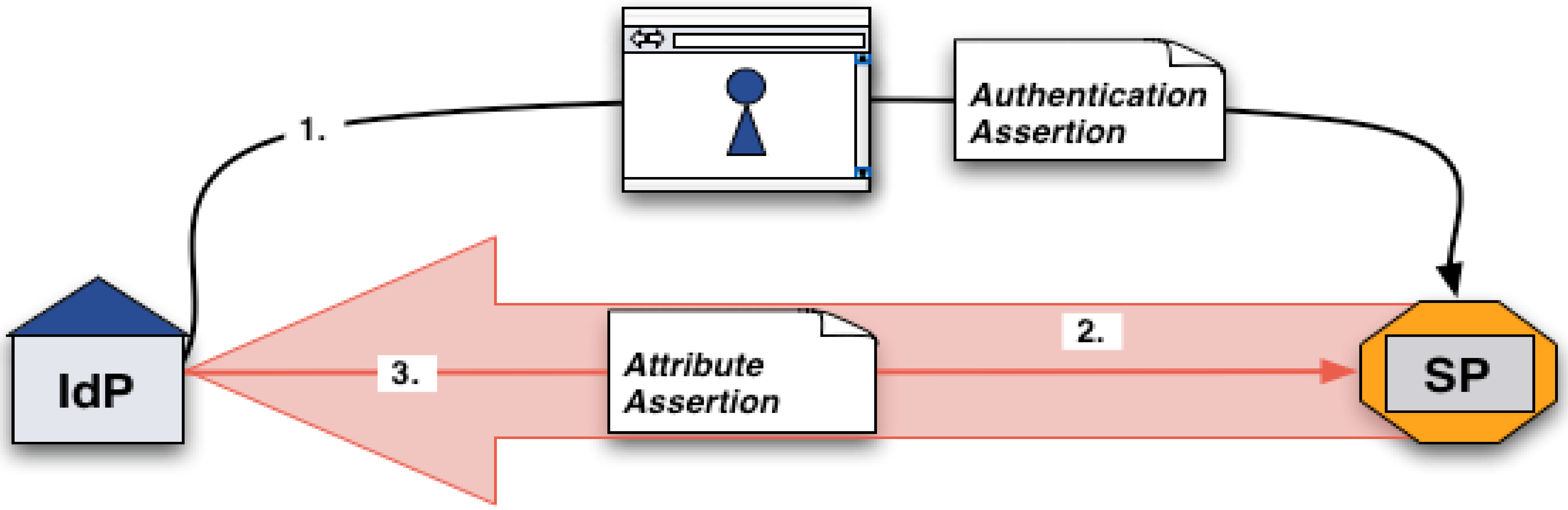
**Goals:**

1. Understand how attributes are transported
2. Learn how attributes are mapped and filtered
3. See how attributes can be used as identifiers
4. Add an attribute mapping and filtering rule

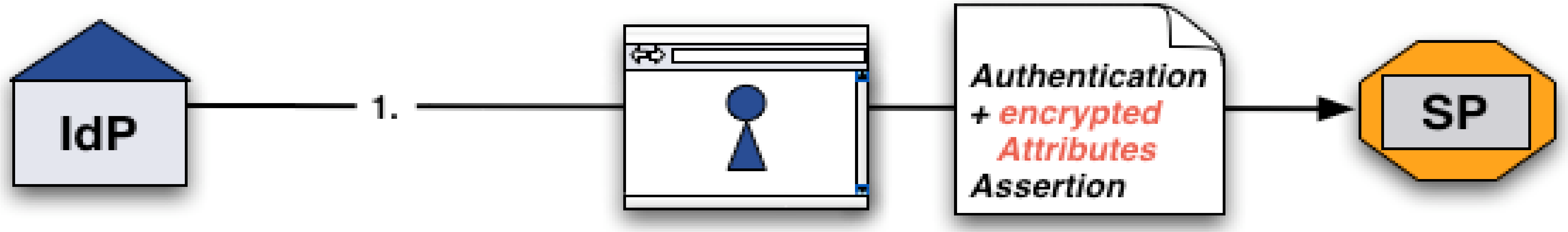
## SP Attribute Terminology

- Push
  - delivering attributes with SSO assertion
- Pull
  - querying for attributes after SSO via back-channel
- Extraction
  - decoding SAML information into neutral data structures mapped to environment or header variables
- Filtering
  - blocking invalid, unexpected, or unauthorized values based on application or community criteria
- Resolution
  - resolving a SSO assertion into a set of additional attributes (e.g. queries)

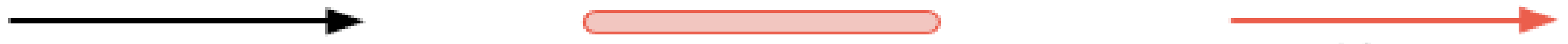
## Attribute Pull vs Push



**Attribute Pull:** Supported by Shibboleth 1.x (default) + Shibboleth 2.x



**Attribute Push:** Supported by Shibboleth 2.x (default)


Web browser HTTP/ HTTPS Connection
Mutually authenticated and encrypted TLS Connection
SOAP

## Attribute Mappings

- SAML attributes from any source are "extracted" using the configuration rules in `/etc/shibboleth/attribute-map.xml`
- Each element is a rule for decoding a SAML attribute and assigning it a local `id` which becomes its mapped variable name
- Attributes can have one or more `id` and multiple attributes can be mapped to the same `id`

## Dissecting an Advanced Attribute Rule

```
<Attribute id="affiliation" aliases="affiliation"
 name="urn:mace:dir:attribute-def:eduPersonScopedAffiliation">
   <AttributeDecoder xsi:type="ScopedAttributeDecoder"
    caseSensitive="false"/>
</Attribute>
```

- `id`
  the primary "id" to map into
- `aliases`
  optional alternate names to map into
- `name`
  SAML attribute name or NameID format to map from
- `AttributeDecoder xsi:type`
  decoder plugin to use (defaults to simple/string)
- `caseSensitive`
  how to compare values at runtime (defaults to true)

## Adding Attribute Mappings

- Add first and last name SAML 2 attribute mappings:

```
$ vim /etc/shibboleth/attribute-map.xml

Line 2:
<Attribute
 name="urn:oid:2.5.4.4" id="sn"/>
<Attribute
 name="urn:oid:2.5.4.42" id="givenName"/>
```

- After saving, changes take effect immediately but NOT for any existing sessions

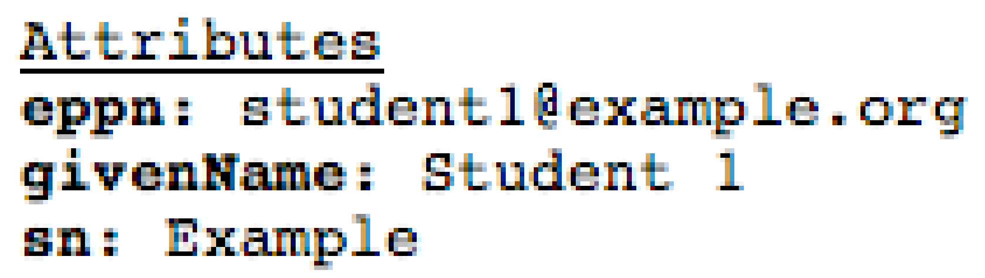
- Therefore close your browser (or delete your session cookies) and continue on next slide …

## Testing Added Attribute Mapping

- Then access `/secure/` again and log in with student1/password. You should get granted access.

- After that, check the Shibboleth Session Handler to see the added attributes are now present:
  `https://spXXX.example.org/Shibboleth.sso/Session`

  You should see the `givenName` and `sn` attribute now as well.



```
Attributes
eppn: student1@example.org
givenName: Student 1
sn: Example
```

## Uncomment All Attribute Mappings

- Let’s uncomment all other attribute mappings.

```
$ vim /etc/shibboleth/attribute-map.xml
```

<u>`Around line 50:`</u>

Remove `<!--`

<u>`Around line 67:`</u>

Remove `-->`

<u>`Around line 69:`</u>

Remove `<!--`

<u>`Around line 122:`</u>

Remove `-->`

- Then logout, go to `/secure/` and access the Session handler
  You now also get `cn` and `primary-affiliation`

## Attributes as Identifiers

- Application must be able to identify user with an attribute
  - Usually used in applications that have a user management e.g. e-learning systems like Moodle, Ilias, Blackboard

- If an attribute is used as an identifier, it should meet some requirements concerning:
  - uniqueness
  - persistence
  - Reassignment - what happens if a student leaves campus?
  - human readability
  - social knowledge

## Common Identifiers

- **local userid/netid/uid** ("login name")
  - usually readable, persistent but not permanent, often reassigned, not unique
- **email address**
  - usually readable, persistent but not permanent, often reassigned, unique
- **eduPersonPrincipalName**
  - usually readable, persistent but not permanent, can be reassigned, unique
- **eduPersonTargetedID / SAML 2.0 persistent ID**
  - not readable, semi-permanent, not reassigned, unique

## REMOTE_USER

- Special single-valued variable that all web applications should support for container-managed authentication of a unique user.
- Any attributes, once extracted/mapped, can be copied to REMOTE_USER.
- Multiple attributes can be examined in order of preference, but only the first value will be used.

## Changing REMOTE_USER

- In case your application needs to have a remote user for authentication, you just could make shibboleth put an attribute (e.g. ”mail”) as REMOTE_USER:

```
/etc/shibboleth/shibboleth2.xml

REMOTE_USER=”mail eppn persistent-id targeted-id"
```

- If mail attribute is available, it will be put into REMOTE_USER
- Attribute `mail` has precedence over `eppn` in this case
- This allows very easy “shibbolization” of some web applications

## Attribute Filtering

- Answers the "who can say what" question on behalf of an application

- Service Provider can make sure that only allowed attributes and values are made available to application

- Usual examples:
  - constraining the possible values or value ranges of an attribute (e.g. eduPersonAffiliation, telephoneNumber, ....)
  - limiting the scopes/domains an IdP can speak for (e.g. OSU cannot assert faculty@umich.edu)
  - limiting custom attributes to particular sources

## Default Filter Policy

- As default, **attributes are filtered out unless there is a rule**
- Shared rule for legal affiliation values
- Shared rule for scoped attributes
- Generic policy applying those rules and letting all other attributes through.
- Check `/var/log/shibboleth/shibd.log` for signs of filtering in case of problems with attributes not being available.
  You would find something like “no rule found, removing all values of attribute (#attribute name#)”

## Add a Source-Based Rule

- Add a rule to limit acceptance of "`sn`" to a single IdP:

```
$ vim /etc/shibboleth/attribute-policy.xml
```

Comment out catch-all section at bottom **and** add surname mapping:

```
Line 55:
<afp:AttributeRule attributeID="sn">
  <afp:PermitValueRule xsi:type="AttributeIssuerString"
         value="https://idpXXX.example.org/idp/shibboleth"/>
</afp:AttributeRule>
<!--
<afp:AttributeRule attributeID="*">
               <afp:PermitValueRule xsi:type="ANY"/>
 </afp:AttributeRule>
-->
```

Then login again `givenName` is filtered out but `sn` is not due to rule.

## Add Catch-all Rule again

- Add a rule to limit acceptance of "`sn`" to a single IdP:

```
$ vim /etc/shibboleth/attribute-policy.xml

Line 56:
<afp:AttributeRule attributeID="sn">
  <afp:PermitValueRule xsi:type="AttributeIssuerString"
         value="https://non.existing.example.org/idp/shibboleth"/>
</afp:AttributeRule>
```

Uncomment catch-all section at bottom:

```
<afp:AttributeRule attributeID="*">
               <afp:PermitValueRule xsi:type="ANY"/>
 </afp:AttributeRule>
```

Then login again `sn` is now filtered out but other attributes aren't anymore.
Because a specific rule exists, the catch-all rule does not apply anymore!

## Remove Specific Rule

- Remove rule for (non-) acceptance of "`sn`":

```
$ vim /etc/shibboleth/attribute-policy.xml
```

  Delete rule for sn (lines 56-58)

```
Line 56:
[Delete rule for sn]
<afp:AttributeRule attributeID="*">
              <afp:PermitValueRule xsi:type="ANY"/>
 </afp:AttributeRule>
```

- Now you should get also `sn` attribute again

## Session Initiation

**Goals:**

1. Learn how to initiate a Shibboleth session
2. Understand their advantages/disadvantages
3. Know where to require a session/what to protect

## Content Protection and Session initiation

- Before access control (will be covered later on) can occur, a Shibboleth session must be initiated
- Session Initiation and content protection go hand in hand
- Requiring a session means the user has to authenticate
- Only authenticated users can access protected content

## Content Protection Settings

https://spaces.internet2.edu/display/SHIB2/NativeSPContentSettings

- Protect hosts, directories, files or queries

- Apache
  - .htaccess (dynamic) or httpd.conf (static)

- Apache / IIS / other
  - RequestMap

    Requires Shibboleth to know exact hostname

    Very powerful and flexible thanks to boolean/regex operations

- Try accessing `https://spXXX.example.org/other-secure/`
  You should get access because the directory is not protected (yet)

  - Create the directory /var/www/html/other-secure on your VM.

## Content protection with .htaccess file

- Let's protect the directory by requiring a Shibboleth session:

```
$ vim /var/www/html/other-secure/.htaccess

AuthType shibboleth
require shibboleth
ShibRequestSetting requireSession 1
```

Last line is the same as old-fashioned 1.3 style:

```
ShibRequireSession On
```

Rules could also be in static httpd configuration directly, see
`/etc/httpd/conf.d/shib.conf` ( default rule for `/secure/` )

## Test Content protection rule

- Clear session and then access `https://spXXX.example.org/other-secure`
  and use student1/password to authentication.
- If you have a typo in the .htaccess file, you might see a 500 error.
- You should be forced to authenticate and then be granted access.
- Right now every user from an Identity Provider configured for your Service Provider gets access
- Content protection with authorization will be covered later

https://spaces.internet2.edu/display/SHIB2/NativeSPApacheConfig

## Content Protection with RequestMap

- `$ vim /var/www/html/other-secure/.htaccess`

```
AuthType shibboleth
require shibboleth
```

- Requiring a session with Request Map:

```
$ vim /etc/shibboleth/shibboleth2.xml

Line 62:
<Host name="spXXX.example.org">
    <Path name=”other-secure" authType="shibboleth"
          requireSession="true"/>
</Host>
```

- mod_shib provides request URL to shibd to proccess it
- Clearing session and then accessing `/other-secure/` now, one also is forced to authenticate

## RequestMap Fragility

- By default, Apache "trusts" the client about what the requested hostname is and reports that value internally

- To illustrate the problem, clear your session and try accessing this URL:

  https://altspXXX.example.org/other-secure

  Script can be accessed unprotected/without a session… ?

- How to fix? Make Apache use “official” ServerName

```
$ vim /etc/httpd/conf/httpd.conf

Line 273:
UseCanonicalName On
ServerName spXXX.example.org
```

## RequestMap Examples

- Accessing `http://spXXX.example.org/other-secure/` (without SSL!) You will stay on http, which may not be secure enough

- Auto-redirecting to SSL using the RequestMap:

```
$ vim /etc/shibboleth/shibboleth2.xml
```

```
Line 62:
<Host name="spXXX.example.org">
     <Path name=”other-secure" authType="shibboleth"
     requireSession="true" redirectToSSL="443"/>
</Host>
```

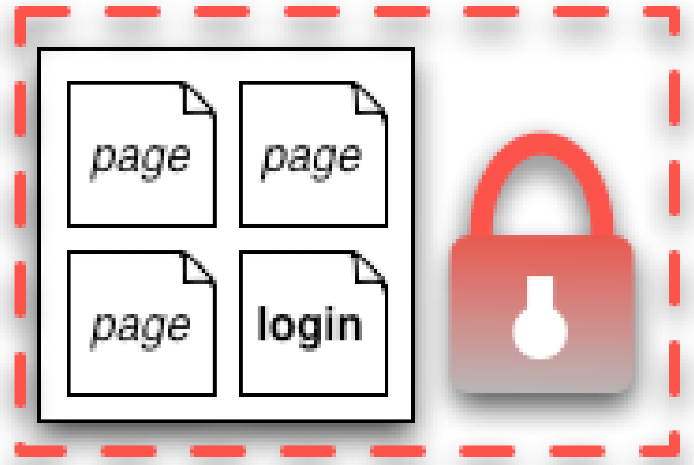
Try again accessing `http://spXXX.example.org/other-secure`
After authentication, you should be on https now!
Same behavior could be achieve with in a .htaccess file. Know how?

## Other Content Settings

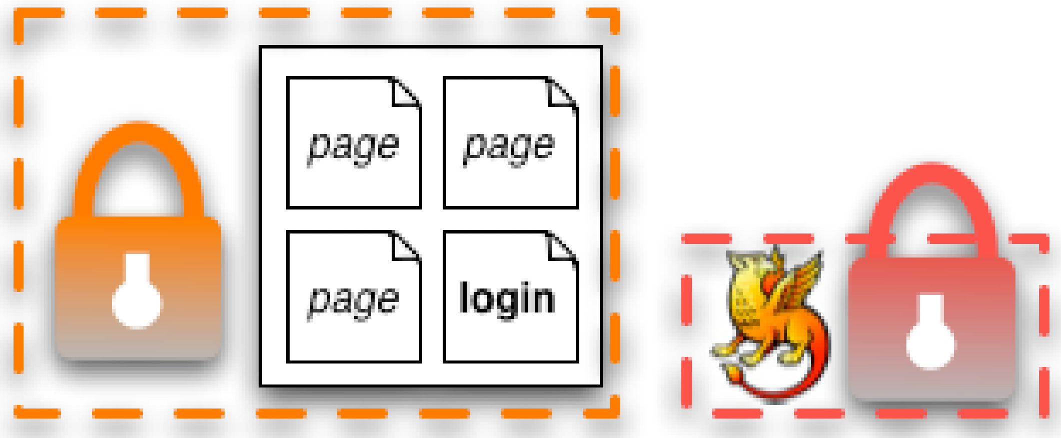
- Requesting types of authentication
  - E.g enforce X.509 user certificate authentication
- Custom error handling pages to use
- Redirection-based error handling
  - In case of an error, redirect user to custom error web page with error message/ type as GET arguments
- **forceAuthn**
  - Disable Single-Sign on and force a re-authentication
- **isPassive**
  - Check whether a user has an SSO session and if he has, automatically create a session on SP without any user interaction
- Supplying a specific IdP to use for authentication

## Where to Require a Shibboleth Session

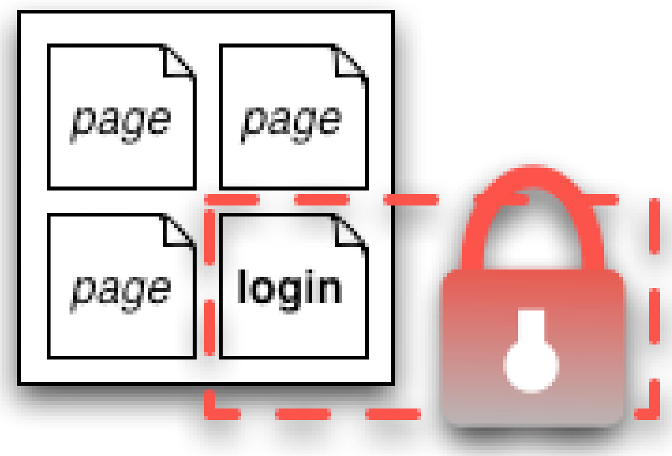
- **Whole application with “required” Shibboleth session**
  - Easiest way to protect a set of documents
  - No dual login possible in general
  - Not user friendly because no “login page” before login
  - Problems with lost HTTP POST requests



- **Whole application with “lazy” Shibboleth session**
  - Well-suited for dual login
  - Lazy sessions are more complicated
  - Authorization can only be done in application



- **Only page that sets up application session**
  - Well-suited for dual login
  - Application can control session time-out
  - **Generally the best solution**



## Protect a Simple Web application

- Access https://spXXX.example.org/cgi-bin/attribute-viewer
  Simple CGI script as a sample application that can be protect
- Lets protect that script with Shibboleth by requiring a session:

```
$ vim /var/www/cgi-bin/.htaccess

AuthType shibboleth
ShibRequireSession on
require shibboleth
```

  This will require a session for all requests to `/cgi-bin/` and make attributes available to application in environment.

- Try again to access script with a browser:
  Script should now show some attributes

## Make Script “see” Shibboleth session

- What if we wanted to grant access also to non-authenticated users but use attributes if somebody is authenticated?
- Use Shibboleth (lazy) session:

```
$ vim /var/www/cgi-bin/.htaccess

AuthType shibboleth
require shibboleth
```

  This will not require a session but make attributes available to application in environment if somebody has a session.

- Try again with a browser:

  https://spXXX.example.org/cgi-bin/attribute-viewer

  Unauthenticated access still possible, but without attributes

## How Script Can Initiate a (Lazy) Session

- Close your browser, and access the attribute-viewer again,
  `https://spXXX.example.org/cgi-bin/attribute-viewer`

- Click on the “Login with Default IdP” button.

  You should be sent to IdP or WAYF and attribute-viewer should display attributes after that.

- Have a look at the HTML source and what it does:
  `https://spXXX.example.org/cgi-bin/attribute-viewer`

- Script initiates Shibboleth session by sending user to:

```
/Shibboleth.sso/Login?target=/cgi-bin/attribute-viewer
/Shibboleth.sso/DS?target=/cgi-bin/attribute-viewer
```

## Try to Initiate a Session Yourself

- You can construct a Session Initiation URL yourself by using these parameters: e.g. supplying the IdP:

```
https://spXXX.example.org/Shibboleth.sso/Login?
  target=https://spXXX.example.org/cgi-bin/attribute-viewer&
  entityID=https://idpXXX.example.org/idp/shibboleth
```

- As you can see, this is the way a Session could be initiated “manually” by an application.

- This allows to create “login links” and therefore replace the WAYF/ Discovery Service

## Session Creation Parameters

https://spaces.internet2.edu/display/SHIB2/NativeSPSessionCreationParameters

- Key Parameters
  - `target` (defaults to homeURL or "/")
  - `entityID` (IdP to use, if known)

- Most parameters can be set at three places.
  In order of precedence:
  - query string parameter to handler
  - a content setting (.htaccess or RequestMap)
  - <SessionInitiator> element

## Lazy Sessions: Some repetition

- Won’t enforce a Shibboleth session but use it if it is available
  - If valid **session exists**
    - then process it as usual (attributes in headers, etc.),

    but if a **session does NOT exist** or is invalid,
    - ignore it and pass on control to application

- Three common cases:
  - Public and private access to the same resources
  - Separation of application and SP session
  - Dual Login (use Shibboleth and some other authentication)

## Using Lazy Sessions

- In place of an API to "doLogin", the SP uses redirects:

  `https://spXXX.example.org/Shibboleth.sso/Login`

- When you/your application want a login to happen, redirect the browser to a SessionInitiator (`/Login` by convention) with any parameters you want to supply

## Access Control

**Goals:**

1. Create some very simple access control rules
2. Get an overview about the three ways to authorize users
3. Understand their advantages/disadvantages

## Access Control

- Integrated  in Service Provider via an AccessControl API built into the request processing flow
- Two implementations are provided by the SP:
  - .htaccess "require" rule processing
  - XML-based policy syntax attached to content via RequestMap
- Third option: Integrate access control into web application

# Access Control Mechanisms

| | 1.a httpd.conf | 1.b .htaccess | 2. XML AccessControl | 3. Application Access Control |
|---|---|---|---|---|
| + | ▪ Easy to configure<br>▪ Can also protect locations or virtual files<br>▪ Regex Support | ▪ Dynamic<br>▪ Easy to configure | ▪ Platform independent<br>▪ Powerful boolean rules<br>▪ Regex Support<br>▪ Dynamic | ▪ Very flexible and powerful with arbitrarily complex rules<br>▪ Regex Support |
| − | ▪ Only works for Apache<br>▪ Not dynamic<br>▪ Not very flexible rules | ▪ Only works for Apache<br>▪ Only works with “real” files and directories | ▪ XML editing<br>▪ Configuration error can prevent SP from restarting | ▪ You have to build it yourself<br>▪ You have to maintain it yourself |

## Aliases

- In the attribute-map.xml file, define something like:

```
<Attribute
 name="urn:mace:dir:attribute-def:eduPersonAffiliation"
 id="Shib-EP-Affiliation"
 aliases="affiliation aff affil">
 […]/>
```

- This allows using rules aliases in authorization rules, e.g.:

```
require affiliation staff
```

instead of:

```
require Shib-EP-Affiliation staff
```

- Aliases can also be used in RequestMap

## 1. Apache httpd.conf or .htaccess files

- Work almost like known Apache “require” rules
  E.g `require affiliation staff`
  or `require mail lukas@testidp.com chad@otheridp.org`

- Special rules:
  - `shibboleth` (no authorization)
  - `valid-user` (require a session, but NOT identity)
  - `user` (REMOTE_USER as usual)
  - `group` (group files as usual)
  - `authnContextClassRef`, `authnContextDeclRef`

- Default is boolean "OR”, use `ShibRequireAll` for AND rule

- Regular expressions supported using special syntax:

```
require rule ~ exp
e.g. require mail ~ ^.*@(it|faculty).example.org$
```

## 1. Example .htaccess File

- Require a user to be a staff member:

```
$ vim /var/www/html/staff-only/.htaccess
```

```
AuthType shibboleth
ShibRequireSession on
require unscoped-affiliation staff
```

Then access : https://spXXX.example.org/staff-only/
with staff1/password. Access should be granted.

- Then try the same with student1/password
  Access should be denied

## 2. More Advanced .htaccess File

- Require a user to be a student or be a member of a group:

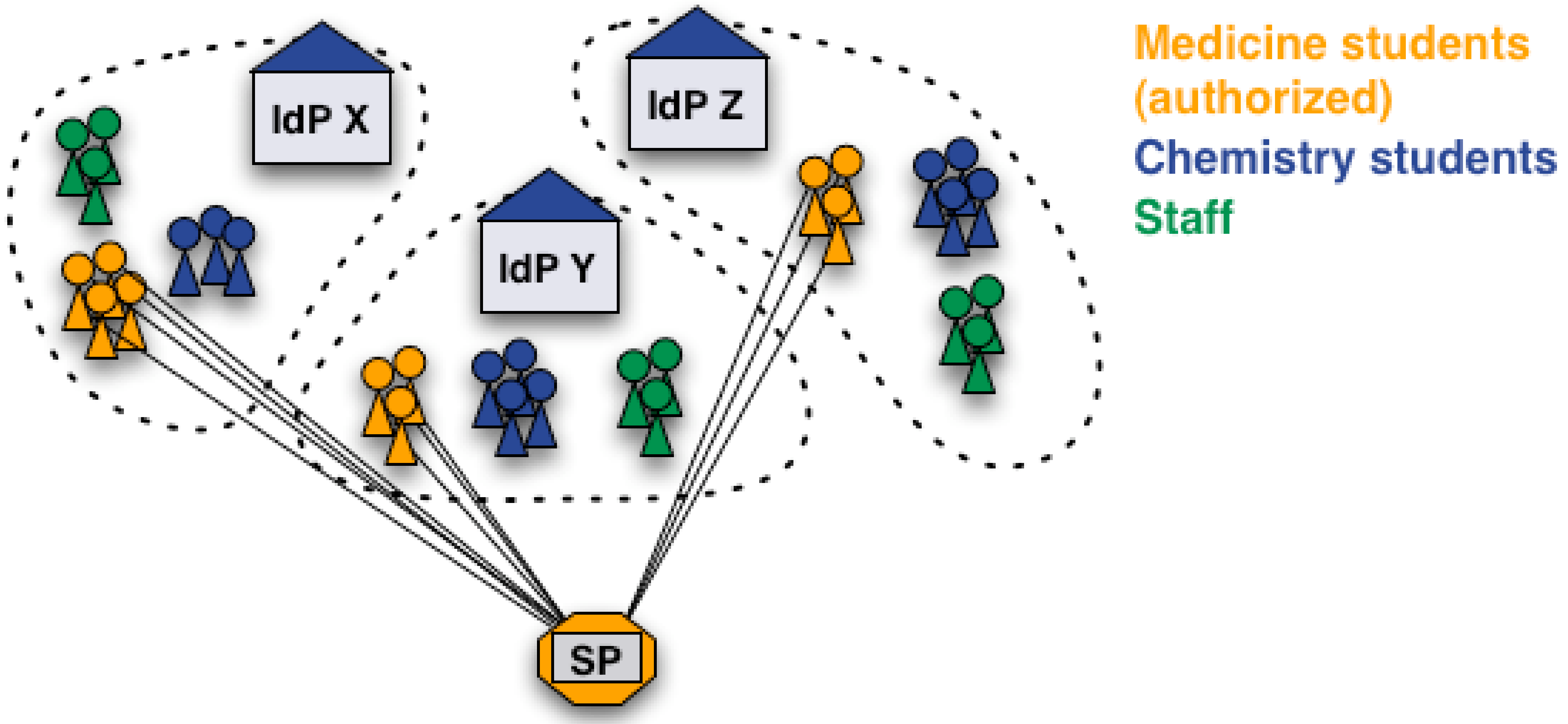
```
$ vim /var/www/html/students-only/.htaccess
```

```
AuthType shibboleth
ShibRequireSession on
require unscoped-affiliation student
require isMemberOf ~ .*photography.*
```

Then access : `https://spXXX.example.org/students-only/` with student1/password. Access should be granted.

- Then try the same with staff1/password
  Access should be granted too because this staff member is a member of urn:example.org:groups:photography!

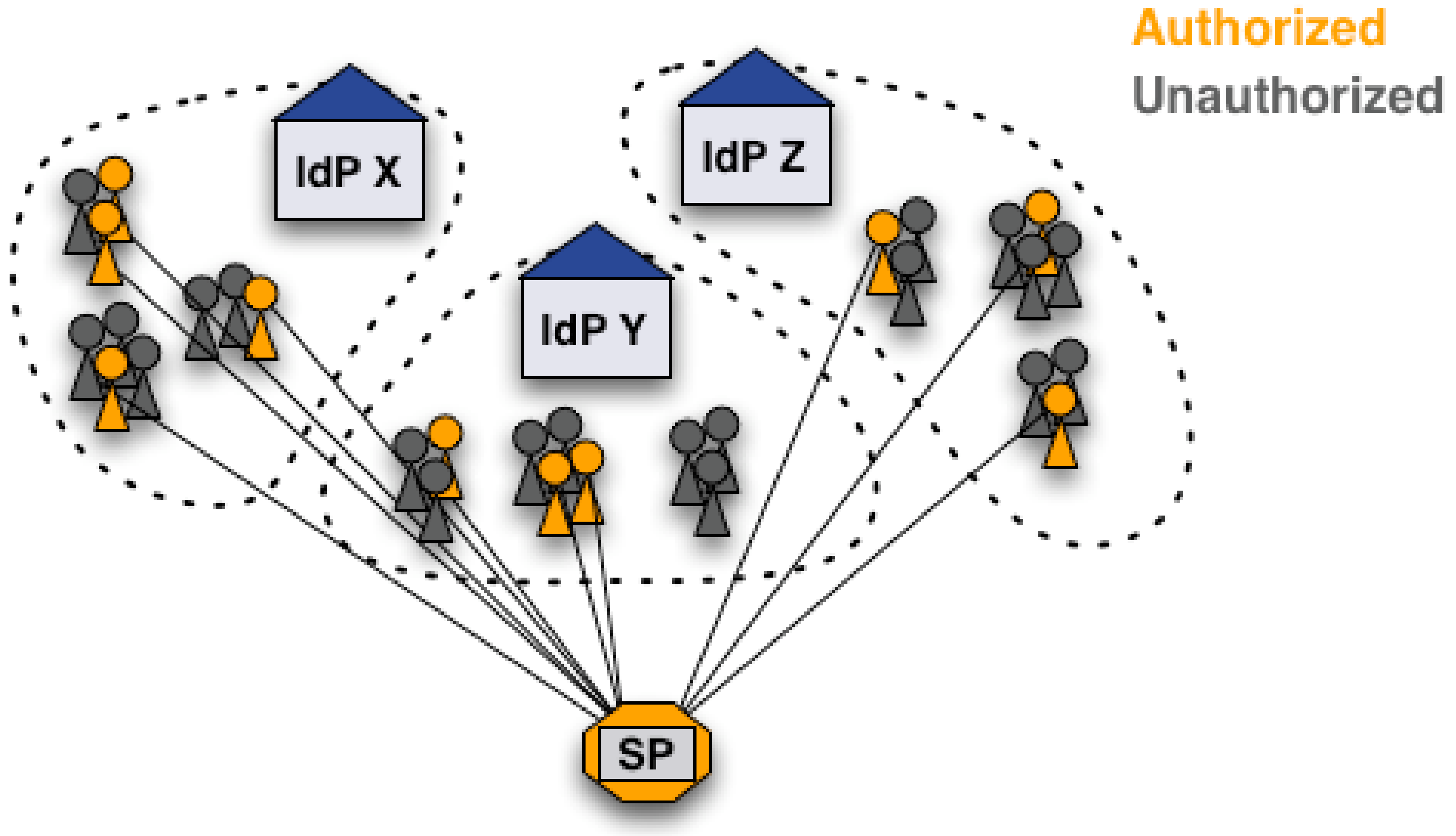
## More complex (real world) example



**Access Rule**

```
ShibRequireAll on
require affiliation student
require homeOrganization idpx.ch idpy.ch idpz.ch
require studyBranch 6200
require studyLevel 15
```

## More likely situation



Multiple solutions on how to authorize only these specific users are demonstrated in the Group Management Tool presentation.

## 2. XML Access Control

- Can be used for access control independent from web server and operating system
- XML Access control rules can be embedded inside RequestMap or can also be dynamically loaded from external file
- Same special rules as .htaccess, adds boolean operators (AND,OR,NOT)

## 2. XML Access Control Example

- Require a group or specific users (same as before):

```
$ vim /etc/shibboleth/shibboleth2.xml
```

```
Line 64:
<Host name="spXXX.example.org">
  <Path name="other-secure" authType="shibboleth" [..]/>
  <Path name="cgi-bin" authType="shibboleth" requireSession="true">
    <AccessControl>
      <OR>
        <RuleRegex require="ismemberof">^.*photography.*$ </RuleRegex>
        <Rule require="unscoped-affiliation">student</Rule>
      </OR>
    </AccessControl>
  </Path>
</Host>
```

Make sure .htaccess file is

```
AuthType shibboleth
require shibboleth
```

- Access https://spXXX.example.org/cgi-bin/attribute-viewer
  Once with alum1 (access denied) and student1 (access granted)

## 3. Application Managed Access Control

- Application uses the Shibboleth attributes from the Apache environment variables or HTTP request headers
- Attributes are then used for access control/authorization

**PHP:**

```
if ($_SERVER['Shib-EP-Affiliation'] == 'staff')
    { grantAccess() }
```

**Perl:**

```
if ($ENV{'Shib-EP-Affiliation'} == 'staff')
    { &grantAccess() }
```

**Java:**

```
if (request.getHeader("Shib-EP-Affiliation").equals("staff") )
   { grantAccess() }
```

## Adding a separate Application

**Goals:**

1. Define another application
2. Protect new application
3. Know how to configure them if necessary

## Terminology

- Service Provider (physical)
  - an installation of the software on a server
- Service Provider/"Resource" (logical)
  - web resources viewed externally as a unit
  - each entityID identifies exactly one logical SP
- SP Application
  - web resources viewed internally as a unit
  - each applicationId identifies exactly one logical application
  - a user session is bound to exactly one application

## Virtualization Concepts

- A single physical SP can host any number of logical SPs
  - A logical SP can then include any number of "applications".
  - Web virtual hosting is often related but is also independent.
  - Applications can inherit or override default configuration settings on a piecemeal basis.
- Multiple physical SPs can also act as a single logical SP (clustering)

## Adding an Application

- Goal: add a second application with its own entityID living on its own virtual host.
- Add the application and map the host to it:

```
$ vim /etc/shibboleth/shibboleth2.xml

Line 62:
<RequestMap applicationId="default">
  <Host name="altspXXX.example.org"
   applicationId="alt"/>

Line 253:
  <ApplicationOverride id="alt" entityID="https://
  altspXXX.example.org/shibboleth"/>
</ApplicationDefaults>
```

## Clustering

- Configure multiple physical installations to share an entityID, and possibly credentials.
- Configuration files often can be identical across servers that share an external hostname.
- Session management:
  - For applications already handling this, 2-3 minute sticky sessions usually sufficient.
  - SP itself now clusterable via ODBC, soon memcached.

## Service Provider Handlers

**Goals:**

1. Understand the idea of a handler
2. Get an overview about the different types of handlers
3. Know how to configure them if necessary

## SP Handlers

- "Virtual" applications inside the SP with API access:
  - SessionInitiator (requests)
    - E.g. `/Shibboleth.sso/Login`
  - AssertionConsumerService (incoming SSO)
    - E.g. `/Shibboleth.sso/SAML/POST`
  - LogoutInitiator (SP signout)
    - E.g. `/Shibboleth.sso/Logout`
  - SingleLogoutService (incoming SLO)
  - ManageNameIDService (adv. SAML)
  - ArtifactResolutionService (adv. SAML)
  - Generic (diagnostics, other useful features)
    - E.g. `/Shibboleth.sso/Session`
    - `/Shibboleth.sso/Status`
    - `/Shibboleth.sso/Metadata`

## SP Handlers

- The URL of a handler = handlerURL + the Location of the handler.
  - e.g. for a virtual host spXXX.example.org with handlerURL of "/Shibboleth.sso", a handler with a Location of "/Login" will be https://spXXX.example.org/Shibboleth.sso/Login
- Go to `https://spXXX.example.org` and see yourself

- Handlers aren't always SSL-only, but usually should be (handlerSSL="true").

- Most of your metadata consists of your entityID, your keys, and your handlers.

- Handlers are never "protected" by the SP
  - But sometimes by IP

## Session Initiators / Discovery

**Goals:**

1. Understand the concepts of discovery/session initiation
2. Chains and protocol precedence
3. Overview about various discovery mechanisms

## Session Initiators / Discovery Concepts

- **Session Initiator**
  Handler that creates a SSO request for an IdP or uses a discovery mechanism to identity the IdP (or both)

- **Discovery (in Shibboleth)**
  Identifying the IdP of a particular user
  Can be internal or external, automatic or invasive

- **WAYF Service**
  Old name in Shibboleth for a particular way to do discovery

- **Handler Chain**
  Sequence of handlers that share configuration and run consecutively until "something useful happens" or an error occurs

## Simplest Case

- Single IdP, **single protocol**, no discovery example:

```
<SessionInitiator
  type="SAML2" id="simple" isDefault="true"
  Location="/Login"
  entityID="https://idpXXX.example.org/idp/shibboleth"
  relayState="cookie"
  defaultACSIndex="1"
  template="bindingTemplate.html"
/>
```

- Resembles the typical approach used in 1.3 SP but omits hardcoding the IdP's location. Location is taken from the metadata, which is useful in case the IdPs service locations change for some reason.

## Intranet Case

- Single IdP, **multiple protocols**, no discovery example:

```
<SessionInitiator type="Chaining" Location="/Login"
         id="Intranet" isDefault="true" relayState="cookie"
         entityID="https://testidp.example.org/idp/shibboleth">
  <SessionInitiator type="SAML2" defaultACSIndex="1"
                       template="bindingTemplate.html"/>
  <SessionInitiator type="Shib1" defaultACSIndex="5"/>
</SessionInitiator>
```

- Protocol precedence controlled by order of SessionInitiators within a chain

- Common properties defined at the top are inherited by SessionInitiators in chain

## Changed Protocol Precedence

- Switch order of chain example:

```
<SessionInitiator type="Chaining" Location="/Login"
          id="Intranet" isDefault="true" relayState="cookie"
          entityID="https://testidp.example.org/idp/shibboleth">
  <SessionInitiator type="Shib1" defaultACSIndex="5"/>
  <SessionInitiator type="SAML2" defaultACSIndex="1"
                       template="bindingTemplate.html"/>
</SessionInitiator>
```

- Still allows either protocol, but if the IdP supports Shibboleth profile of SAML 1, it will be used instead

## Identity Provider Discovery

- Protocol SessionInitiators work when the IdP is known
- For consistency, discovery is implemented with alternate SessionInitiators that operate only when the IdP is NOT known.
- A typical federated chain includes one or more "protocol" handlers followed by a single "discovery" handler at the end

## Typical Discovery Methods

- **External Options:**
  - Older WAYF model, specific to Shibboleth/SAML1, SP loses control if a problem occurs
  - Newer SAMLDS model, recently standardized, supports multiple SSO protocols and allows the SP to control the process

- **Internal Options**
  - Implemented by an application
  - Followed by a redirect with the entityID. E.g `/Shibboleth.sso/Login ?entityID=https://idpXXX.example.org/idp/shiboleth`
  - Advanced "Cookie", "Form", and "Transform" SessionInitiators

## Discovery Service Case (very common)

- Multiple protocols, discovery via DS:

```
<SessionInitiator type="Chaining" Location="/DS"
 id="DS" isDefault="true" relayState="cookie">
  <SessionInitiator type="SAML2"
    defaultACSIndex="1" acsByIndex="false"
    template="bindingTemplate.html"/>
  <SessionInitiator type="Shib1" defaultACSIndex="5"/>
  <SessionInitiator type="SAMLDS"
   URL="https://ds.example.org/DS/WAYF"/>
</SessionInitiator>
```

- Same as intranet case, but omits entityID and adds a "safety net" at the bottom
- Last SessionInitiator in chain tells the DS to return the user to this location with a lazy session redirect that will invoke an earlier handler (SAML2 or Shib1) in the chain

## External Discovery/WAYF

⊕
- Easy to use (you don't have to set up something)
- Easier generation of statistical data if DS is centralized

⊖
- Loss of control, UI fidelity
- Impact of errors
- Choice of IdPs becomes arbitrary: metadata errors have to be handled
- Comprehensiveness is impossible in a world of multiple federations, and the list can already become too long if there's only one federation

## Advanced SessionInitiators

- **Form SessionInitiator**
  - Crude DS relying on HTML form template, NOT meant to replace the actual DS implementation
- **Cookie SessionInitiator** (early in chain)
  - Parses _saml_idp cookie maintained by idpHistory feature and sets entityID ahead of other handlers
- **Transform SessionInitiator** (early in chain)
  - Applies substitution or regex patterns against entityID to turn it into another entityID until metadata is available

## Advanced URL Discovery Example

- Use a form to prompt for the entityID, or a domain name or email address that is input to a convention (https://idp.domain/idp/shibboleth):

```
<SessionInitiator type="Chaining" Location="/Login" isDefault="false"
   id="Intranet" relayState="cookie" entityID="https://idpXXX.example.org/idp/shibboleth">
   <SessionInitiator type="SAML2" defaultACSIndex="1" template="bindingTemplate.html"/>
   <SessionInitiator type="Shib1" defaultACSIndex="5"/>
</SessionInitiator>
....
<SessionInitiator type="Chaining" Location="/DS" id="DS" relayState="cookie" isDefault="true">
   <SessionInitiator type="Transform">
               <Subst>https://idpXXX.$entityID/idp/shibboleth</Subst>
               <Regex match=".+@(.+)">https://idpXXX.$1/idp/shibboleth</Regex>
   </SessionInitiator>
   <SessionInitiator type="SAML2" defaultACSIndex="1" acsByIndex="false"
   template="bindingTemplate.html"/>
   <SessionInitiator type="Shib1" defaultACSIndex="5"/>
   <!-- Comment out SAMLDS
   <SessionInitiator type="SAMLDS" […] />
   -->
   <SessionInitiator type="Form" template="discoveryTemplate.html"/>
</SessionInitiator>
```

- When asked enter e.g. “john.doe@example.org” or just “example.org”