WebSphere Liberty Security Introduction

James Mulvey

STSM, WAS Security and Z architect

jmulvey@us.ibm.com

.3

March 2018

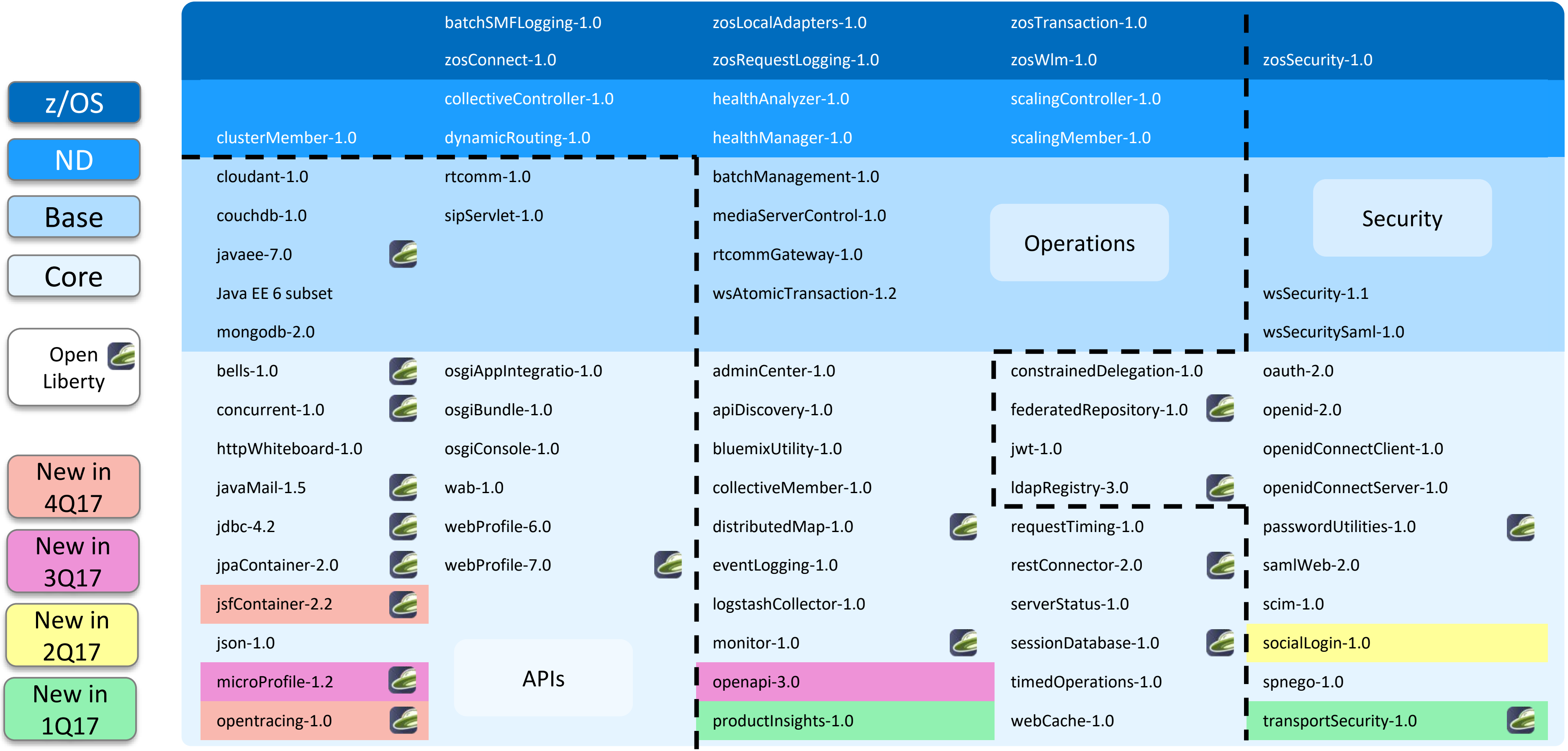
Agenda

- Security features in Liberty
  - Review of feature appSecurity-2.0
  - Comparing Liberty security with traditional WAS
- Authentication
  - User registries (URs), SCIM
  - Web applications security
- Single sign-on overview
  - LTPA, SAML, SPNEGO, OAuth 2.0, JWTs, OpenID Connect, and SocialLogin
- Authorization
  - JEE groups and roles in Liberty
- Other items
  - securityUtility
  - Java EE 8/JSR 375, Auditing and Liberty (beta)
  - What we’re working on now

Reminder: Part 2 for Liberty security

- **Security Hardening and Best Practices for Running WebSphere Liberty in Production - April 6th at 11am ET**
- In this session, we'll cover some fundamental best practices related to Liberty and security in a production environment. http://ibm.biz/BdZEwB

Liberty Current Features 17.0.0.4



z/OS

ND

Base

Core

Open Liberty

New in 4Q17

New in 3Q17

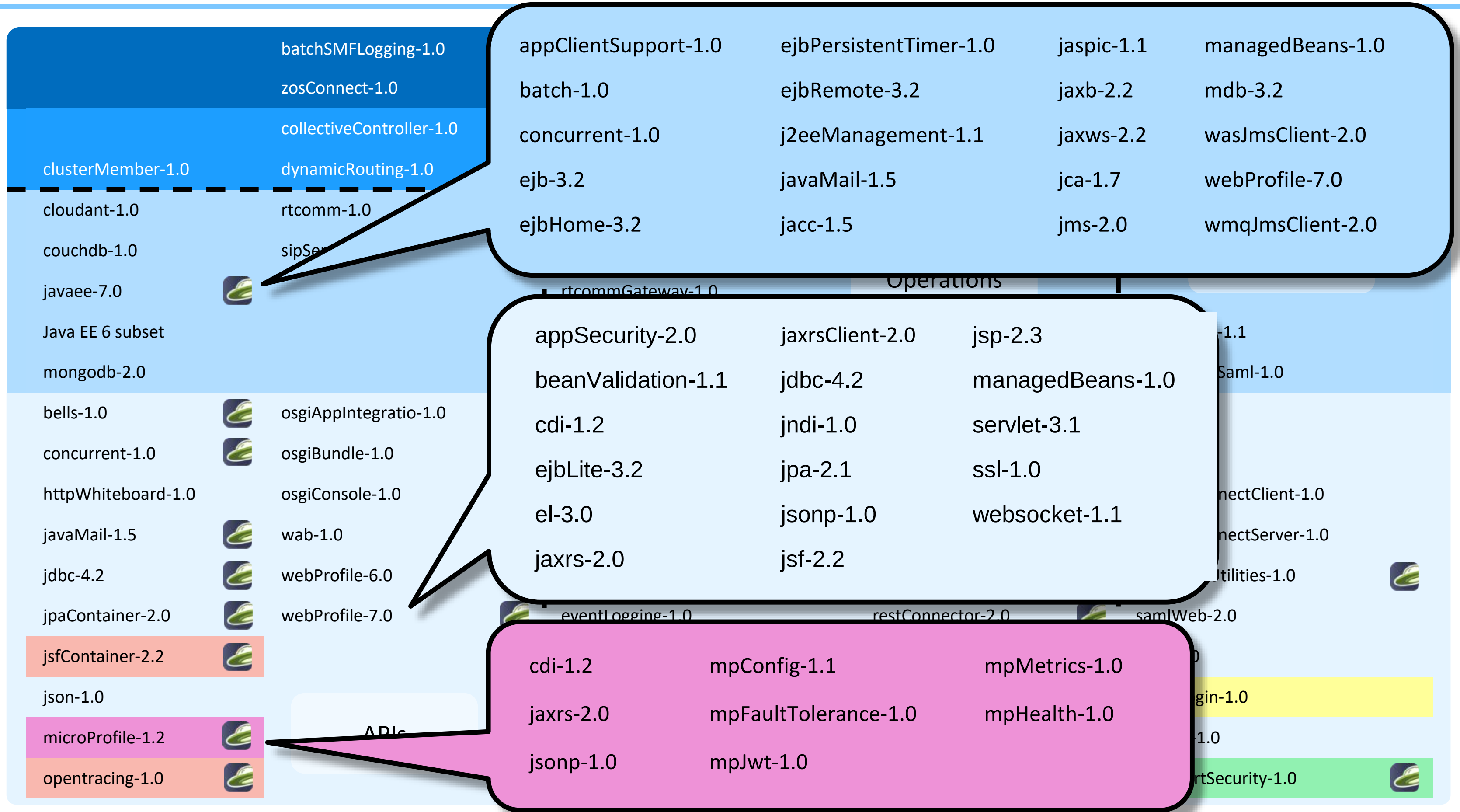
New in 2Q17

New in 1Q17

	batchSMFLogging-1.0	zosLocalAdapters-1.0	zosTransaction-1.0	
	zosConnect-1.0	zosRequestLogging-1.0	zosWlm-1.0	zosSecurity-1.0
	collectiveController-1.0	healthAnalyzer-1.0	scalingController-1.0	
clusterMember-1.0	dynamicRouting-1.0	healthManager-1.0	scalingMember-1.0	
cloudant-1.0	rtcomm-1.0	batchManagement-1.0		
couchdb-1.0	sipServlet-1.0	mediaServerControl-1.0	Operations	Security
javaee-7.0		rtcommGateway-1.0		
Java EE 6 subset		wsAtomicTransaction-1.2		wsSecurity-1.1
mongodb-2.0				wsSecuritySaml-1.0
bells-1.0	osgiAppIntegratio-1.0	adminCenter-1.0	constrainedDelegation-1.0	oauth-2.0
concurrent-1.0	osgiBundle-1.0	apiDiscovery-1.0	federatedRepository-1.0	openid-2.0
httpWhiteboard-1.0	osgiConsole-1.0	bluemixUtility-1.0	jwt-1.0	openidConnectClient-1.0
javaMail-1.5	wab-1.0	collectiveMember-1.0	ldapRegistry-3.0	openidConnectServer-1.0
jdbc-4.2	webProfile-6.0	distributedMap-1.0	requestTiming-1.0	passwordUtilities-1.0
jpaContainer-2.0	webProfile-7.0	eventLogging-1.0	restConnector-2.0	samlWeb-2.0
jsfContainer-2.2		logstashCollector-1.0	serverStatus-1.0	scim-1.0
json-1.0		monitor-1.0	sessionDatabase-1.0	socialLogin-1.0
microProfile-1.2	APIs	openapi-3.0	timedOperations-1.0	spnego-1.0
opentracing-1.0		productInsights-1.0	webCache-1.0	transportSecurity-1.0

Liberty Current Features 17.0.0.4

- z/OS
- ND
- Base
- Core
- Open Liberty
- New in 4Q17
- New in 3Q17
- New in 2Q17
- New in 1Q17



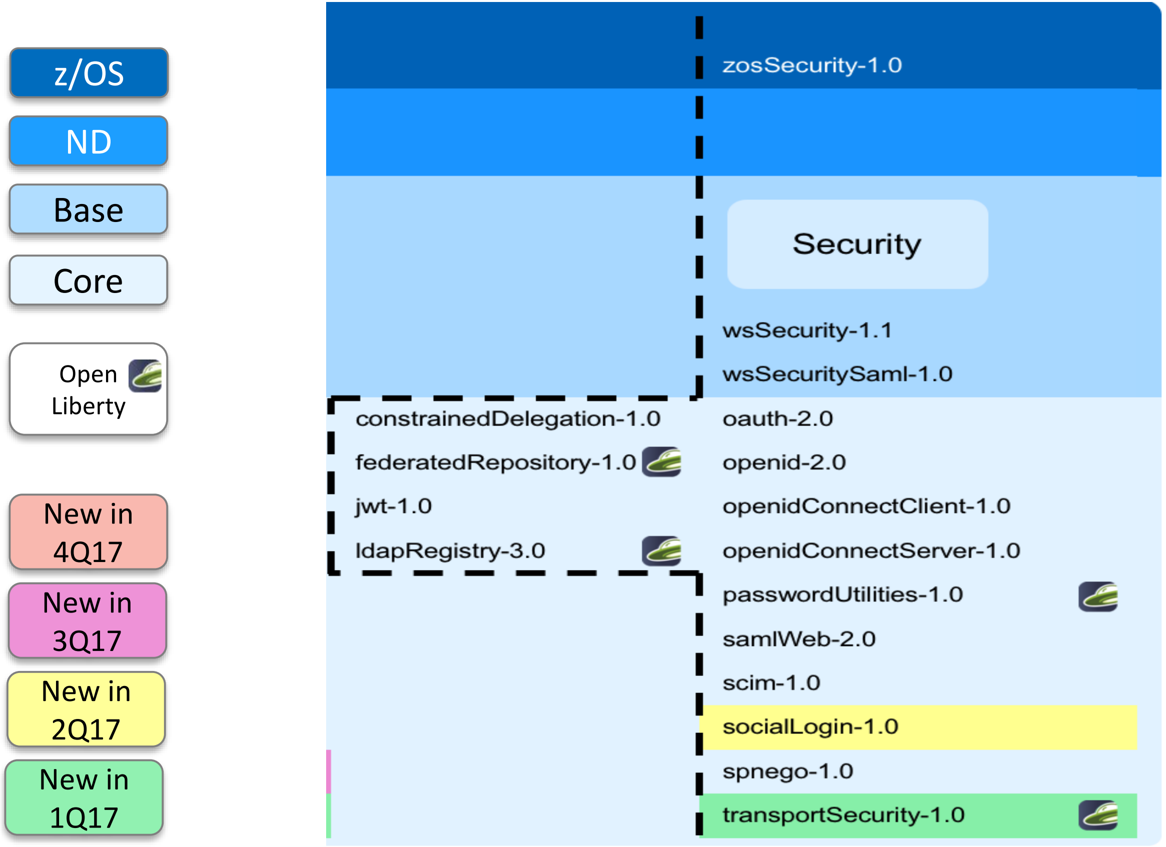
batchSMFLogging-1.0
zosConnect-1.0
collectiveController-1.0
clusterMember-1.0
dynamicRouting-1.0
cloudant-1.0
rtcomm-1.0
couchdb-1.0
sipSe
javaee-7.0
Java EE 6 subset
mongodb-2.0
bells-1.0
osgiAppIntegratio-1.0
concurrent-1.0
osgiBundle-1.0
httpWhiteboard-1.0
osgiConsole-1.0
javaMail-1.5
wab-1.0
jdbc-4.2
webProfile-6.0
jpaContainer-2.0
webProfile-7.0
jsfContainer-2.2
json-1.0
microProfile-1.2
opentracing-1.0
APIs
rtcommGateway-1.0
Operations
-1.1
Saml-1.0
nectClient-1.0
nectServer-1.0
tilities-1.0
eventLogging-1.0
restConnector-2.0
samlWeb-2.0
0
gin-1.0
1.0
rtSecurity-1.0

appClientSupport-1.0	ejbPersistentTimer-1.0	jaspic-1.1	managedBeans-1.0
batch-1.0	ejbRemote-3.2	jaxb-2.2	mdb-3.2
concurrent-1.0	j2eeManagement-1.1	jaxws-2.2	wasJmsClient-2.0
ejb-3.2	javaMail-1.5	jca-1.7	webProfile-7.0
ejbHome-3.2	jacc-1.5	jms-2.0	wmqJmsClient-2.0

appSecurity-2.0	jaxrsClient-2.0	jsp-2.3
beanValidation-1.1	jdbc-4.2	managedBeans-1.0
cdi-1.2	jndi-1.0	servlet-3.1
ejbLite-3.2	jpa-2.1	ssl-1.0
el-3.0	jsonp-1.0	websocket-1.1
jaxrs-2.0	jsf-2.2	

cdi-1.2	mpConfig-1.1	mpMetrics-1.0
jaxrs-2.0	mpFaultTolerance-1.0	mpHealth-1.0
jsonp-1.0	mpJwt-1.0	

Liberty Current Features 17.0.0.4



z/OS

ND

Base

Core

Open Liberty

New in 4Q17

New in 3Q17

New in 2Q17

New in 1Q17

zosSecurity-1.0

Security

wsSecurity-1.1

wsSecuritySaml-1.0

constrainedDelegation-1.0

federatedRepository-1.0

jwt-1.0

ldapRegistry-3.0

oauth-2.0

openid-2.0

openidConnectClient-1.0

openidConnectServer-1.0

passwordUtilities-1.0

samlWeb-2.0

scim-1.0

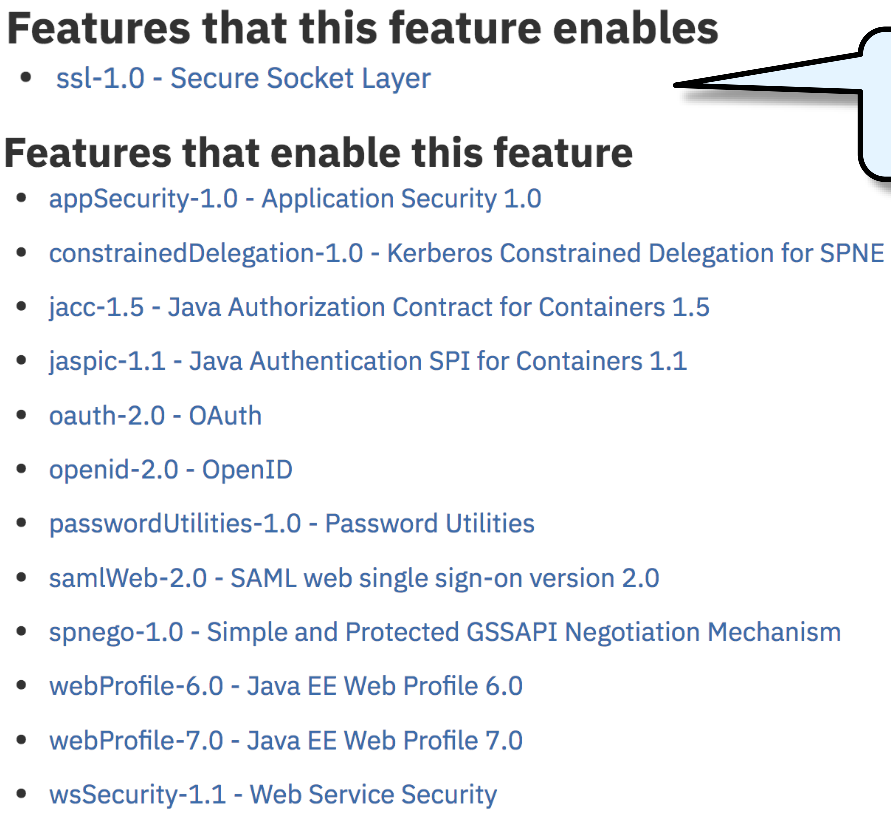
socialLogin-1.0

spnego-1.0

transportSecurity-1.0

appSecurity-2.0

```
<feature>appSecurity-2.0</feature>
```



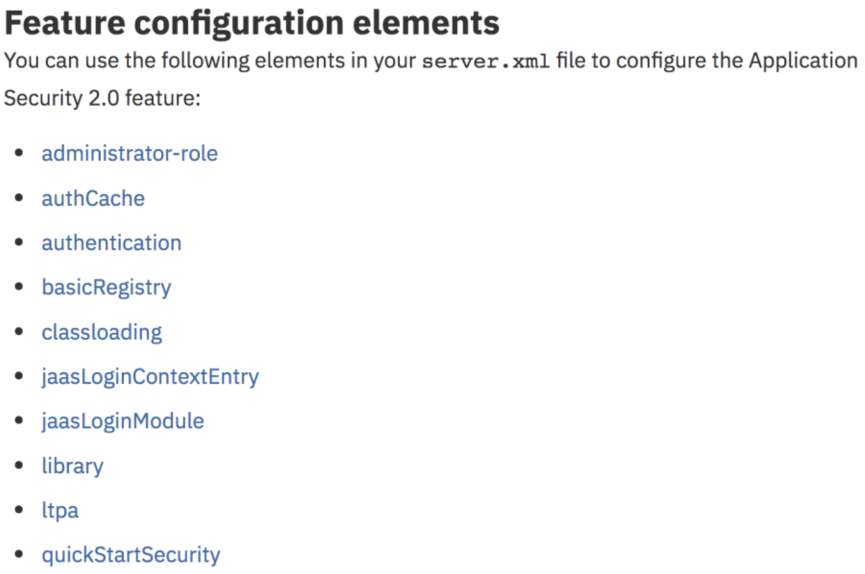
Features that this feature enables

- ssl-1.0 - Secure Socket Layer

Features that enable this feature

- appSecurity-1.0 - Application Security 1.0
- constrainedDelegation-1.0 - Kerberos Constrained Delegation for SPNE
- jacc-1.5 - Java Authorization Contract for Containers 1.5
- jaspic-1.1 - Java Authentication SPI for Containers 1.1
- oauth-2.0 - OAuth
- openid-2.0 - OpenID
- passwordUtilities-1.0 - Password Utilities
- samlWeb-2.0 - SAML web single sign-on version 2.0
- spnego-1.0 - Simple and Protected GSSAPI Negotiation Mechanism
- webProfile-6.0 - Java EE Web Profile 6.0
- webProfile-7.0 - Java EE Web Profile 7.0
- wsSecurity-1.1 - Web Service Security

Enabling appSecurity-2.0 means enabling SSL/TLS for the server. It also means LTPA is enabled. More on that coming …



Feature configuration elements

You can use the following elements in your `server.xml` file to configure the Application Security 2.0 feature:

- administrator-role
- authCache
- authentication
- basicRegistry
- classloading
- jaasLoginContextEntry
- jaasLoginModule
- library
- ltpa
- quickStartSecurity

- openidConnectClient-1.0, openidConnectProvider-1.0 – Open ID Connect Client and Provider

Comparing Liberty security to traditional WAS

	Liberty	Traditional WAS
Minimal ports opened	Yes	No
File user registry	Yes (server.xml)	Yes (file based)
Federated user registries	Yes	Yes
OAuth, OpenID, OIDC client	Yes	Yes
OpenID Connect Provider	Yes	No
LTPA, SPNEGO tokens	Yes	Yes
SAML Web SSO	Yes	Yes
JSON web tokens (JWTs)	Yes	No
Secure remote admin	Yes (mandatory)	Yes (but can be turned off)
User and group API	Yes	Yes
Auditing	No (in beta)	Yes
Local OS registry	Yes (z/OS SAF only)	Yes

Authentication

User Registries supported with Liberty

March 2018

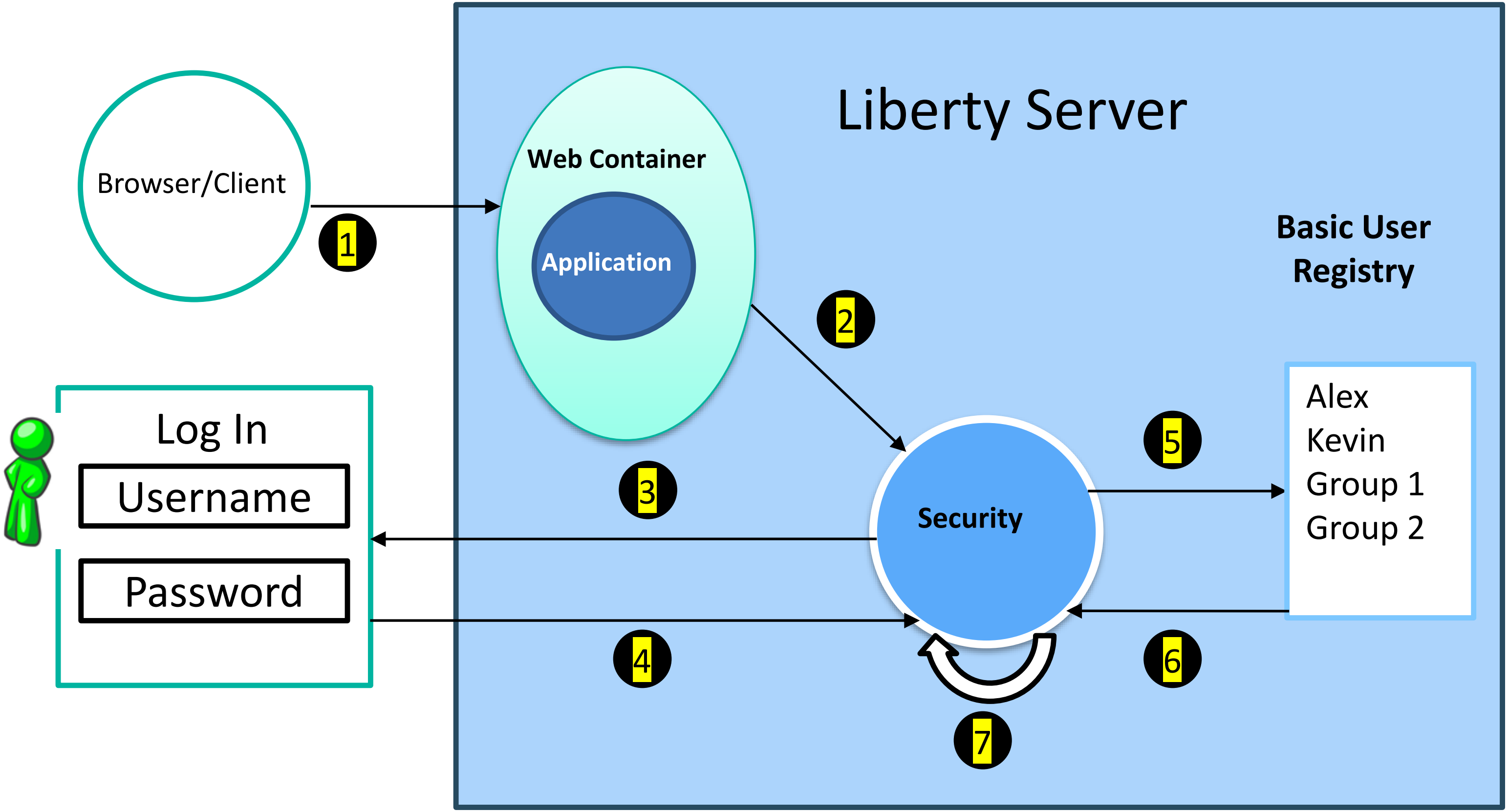
User Registries

- User Registries (UR) contain the user and group information.
- They are used during the authentication process.
- When a user provides their credentials (ie: username and password), they are checked against the UR that is configured.
- They also contain the group information which is later used for authorization checks to permit/deny access to protected resources.
- Liberty profile supports multiple types of user registries.

Types of registries in Liberty

- Basic
- Quickstart
- LDAP (Lightweight Directory Access Protocol)
- Custom
- Local (z/OS only – SAF)
- Federated

Basic user registry



1. Access protected resource
2. Web Container calls security runtime
3. Security runtime prompts user for credentials
4. User enters credentials
5. Security runtime checks credentials against the **user registry in the Liberty server config**
6. Registry returns if credentials are valid or not
7. If valid, proceed to authorization

Basic user registry

- A basic registry is a simple file based registry.
- The user and group information is configured in the server.xml
- It is typically used in development environments (not production)
- It is a fast and simple way to test the access to your protected application.
- Example

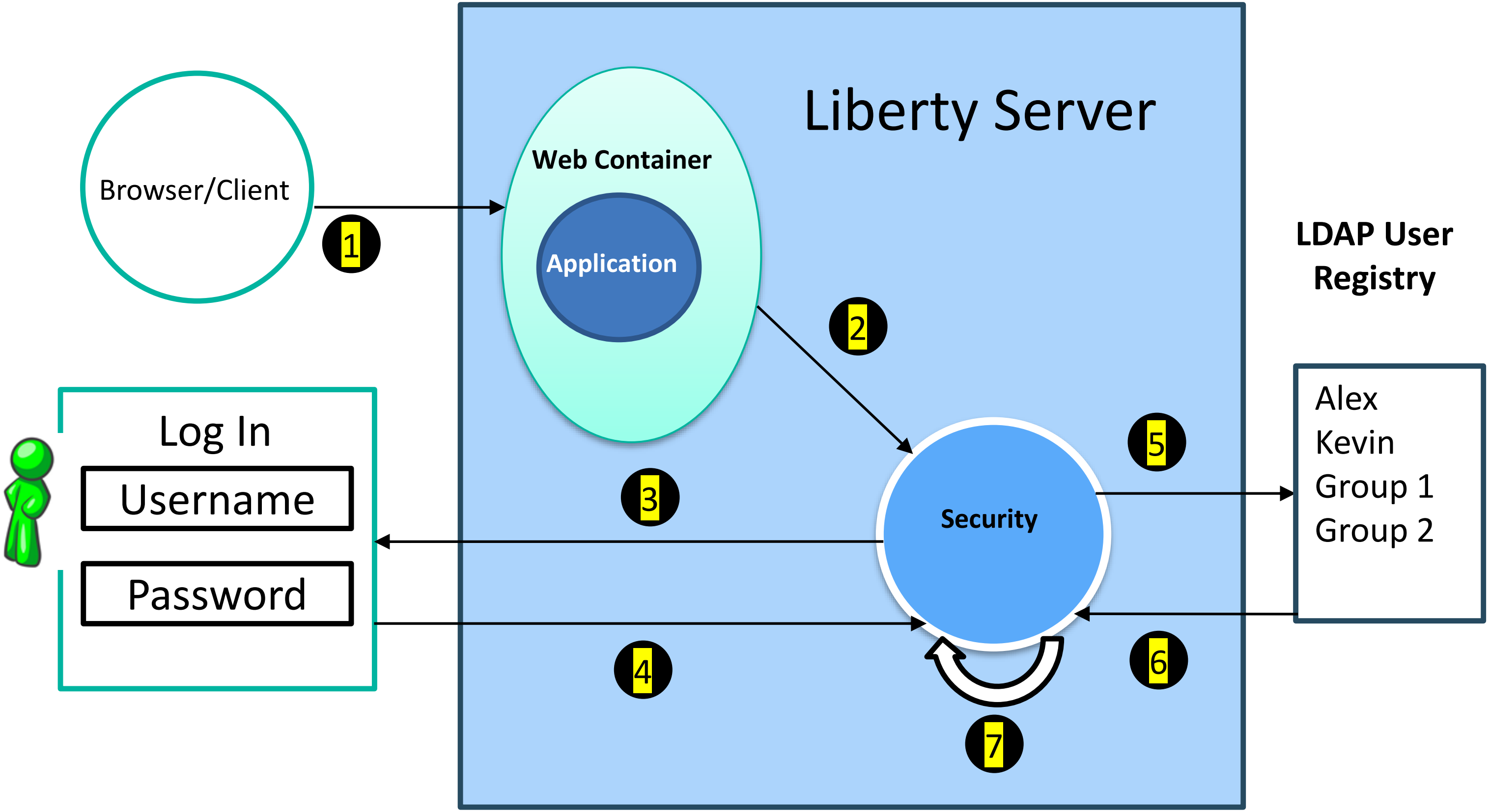
```
<basicRegistry id="basic1" realm="SampleRealm">
        <user name="user2" password="user2pwd" />
        <user name="user1" password="user1pwd" />
        <group name="group1">
           <member name="user1" />
        </group>
</basicRegistry>
```

Quickstart security

- Quickstart security is a form of basic registry. It’s designed to provide a basic security configuration with minimal complexity. Not for production use.
- It contains just one user.
- This user also has access to the admin role
- Used primarily to provide protection for remote operations (restConnector) when no application security is needed.
- Example

```
<quickStartSecurity userName="bob” userPassword="bobpassword"/>
```

LDAP user registry



1. Access protected resource
2. Web Container calls security runtime
3. Security runtime prompts user for credentials
4. User enters credentials
5. Security runtime checks credentials against the **user registry from an LDAP server**
6. Registry returns if credentials are valid or not
7. If valid, proceed to authorization

LDAP user registries

- LDAP registries are based on the LDAP protocol.
- It is a very well known industry standard protocol used by many customers.
- There are many well known LDAP servers in the market. Examples include:
  - IBM Directory Server
  - Active Directory
- The latest LDAP version is v3.
- Liberty profile supports any LDAP server compliant to LDAP v3.
- Most customers use LDAP in production.

https://www.ibm.com/support/knowledgecenter/en/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_sec_ldap.html

LDAP Configuration

- Here is an example of LDAP configuration in Liberty (IBM Directory Server):

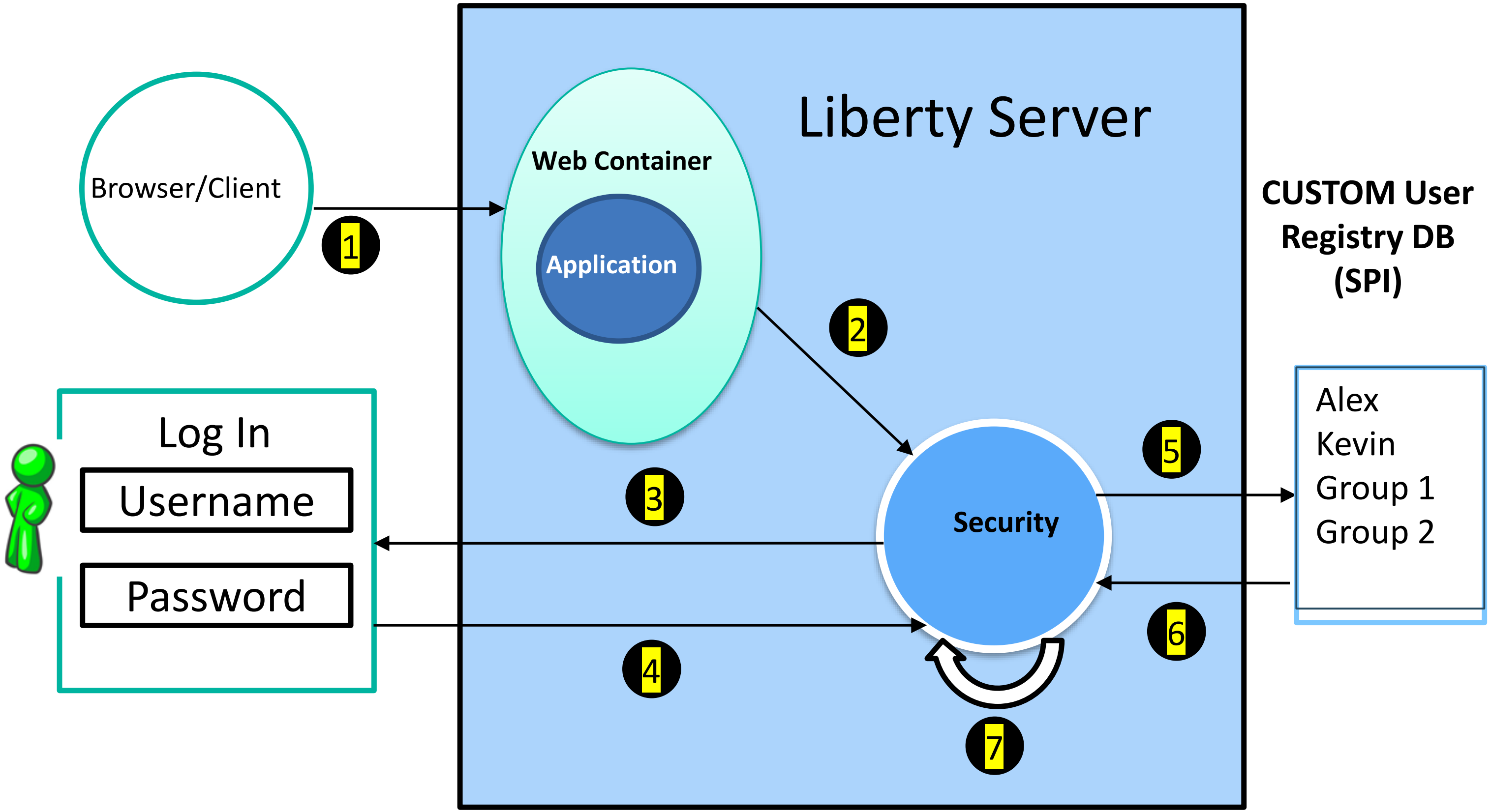
```
<ldapRegistry id="ldap"
        realm="SampleLdapIDSRealm"
        host=“myHost.com" port=“443“
        ignoreCase="true"
        baseDN="o=ibm,c=us"
        bindDN=“o=myCompany"
        bindPassword=“mySecret"
        ldapType="IBM Tivoli Directory Server"
        idsFilters="ibm_dir_server"
        sslEnabled="true"
        sslRef="LDAPSSLSettings">
</ldapRegistry>
```

LDAP Configuration

- Here is an example of LDAP configuration in Liberty showing use of filtering for IBM Directory Server :

```
<idsLdapFilterProperties id="ibm_dir_server"

        userFilter="(&amp;(uid=%v)(objectclass=ePerson))"

        groupFilter="(&amp;(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)(objectclass=groupOfURLs)))"

        userIdMap="*:uid"

        groupIdMap="*:cn"

        groupMemberIdMap="ibm-allGroups:member;ibm-allGroups:uniqueMember;groupOfNames:member;groupOfUniqueNames:uniqueMember">

</idsLdapFilterProperties>
```

Custom user registry (CUR)



1. Access protected resource
2. Web Container calls security runtime
3. Security runtime prompts user for credentials
4. User enters credentials
5. Security runtime checks credentials against the **CUSTOM user registry (SPI implementation can be used)**
6. Registry returns if credentials are valid or not
7. If valid, proceed to authorization

Custom User Registry

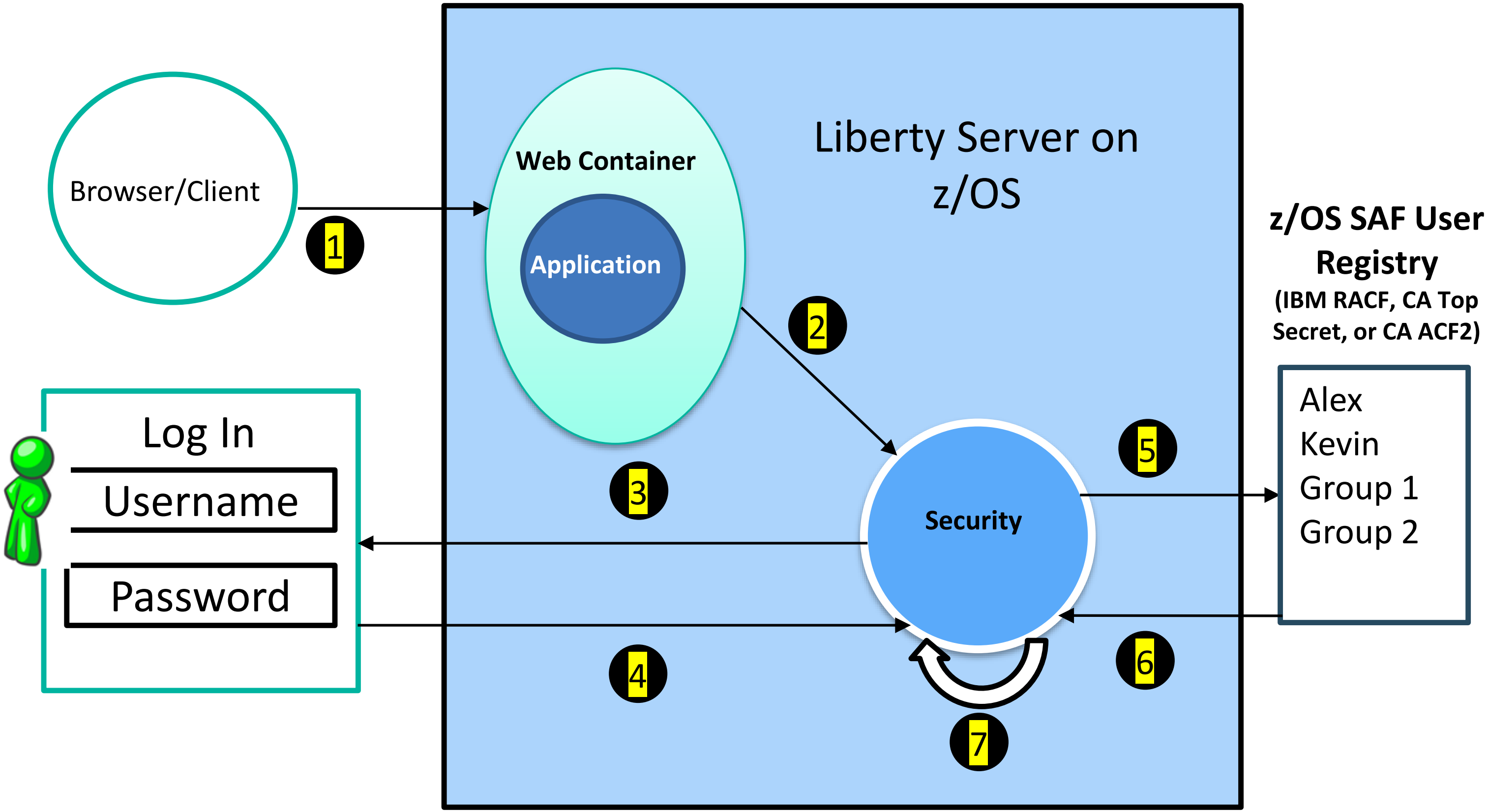
- Liberty profile provides an SPI (service provider interface) (com.ibm.websphere.security.UserRegistry) where customers can provide their own User Registry implementation.
  - Also need to implement to com.ibm.websphere.security.Result interface.
  - The SPI helps with integrating with any type of user repository. For example – a database based repository
- Customers implement this and configure this as user extension feature in the server.xml.

```
<featureManager>
   ... <feature>usr:customRegistrySample-1.0</feature>
   </featureManager>
```

https://www.ibm.com/support/knowledgecenter/en/SSEQTP_8.5.5/com.ibm.websphere.wlp.doc/ae/twlp_sec_custmr.html

- This is one of the common ways to integrate with existing user customized repositories.

z/OS local registry (SAF)



1. Access protected resource
2. Web Container calls security runtime
3. Security runtime prompts user for credentials
4. User enters credentials
5. Security runtime checks credentials against the **SAF user registry**
6. Registry returns if credentials are valid or not
7. If valid, proceed to authorization

Local/SAF User Registry

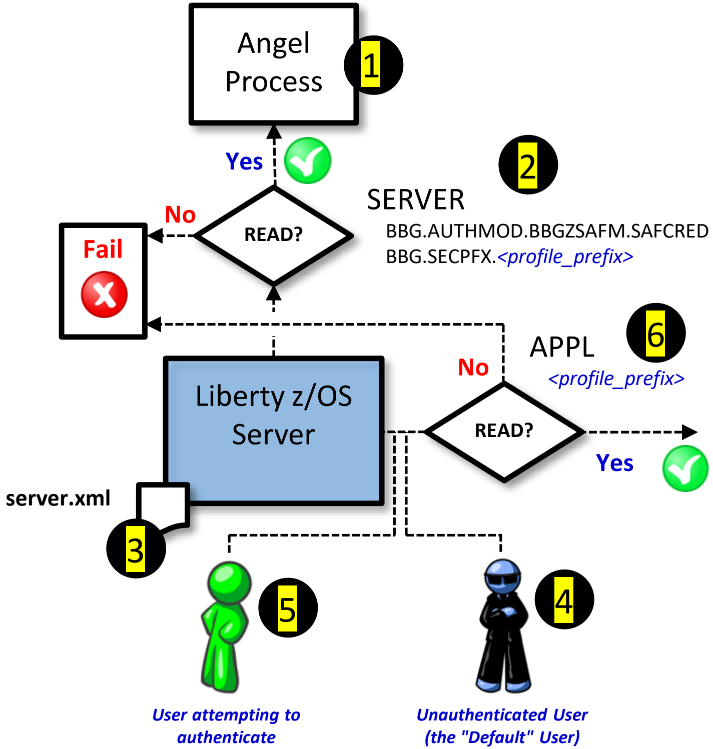
- Liberty profile provides support for the System Authorization Facility (SAF) framework on z/OS.
  - Need to configure zosSecurity-1.0 in featureManager

```
<feature>zosSecurity-1.0</feature>
```

- Example server.xml config:

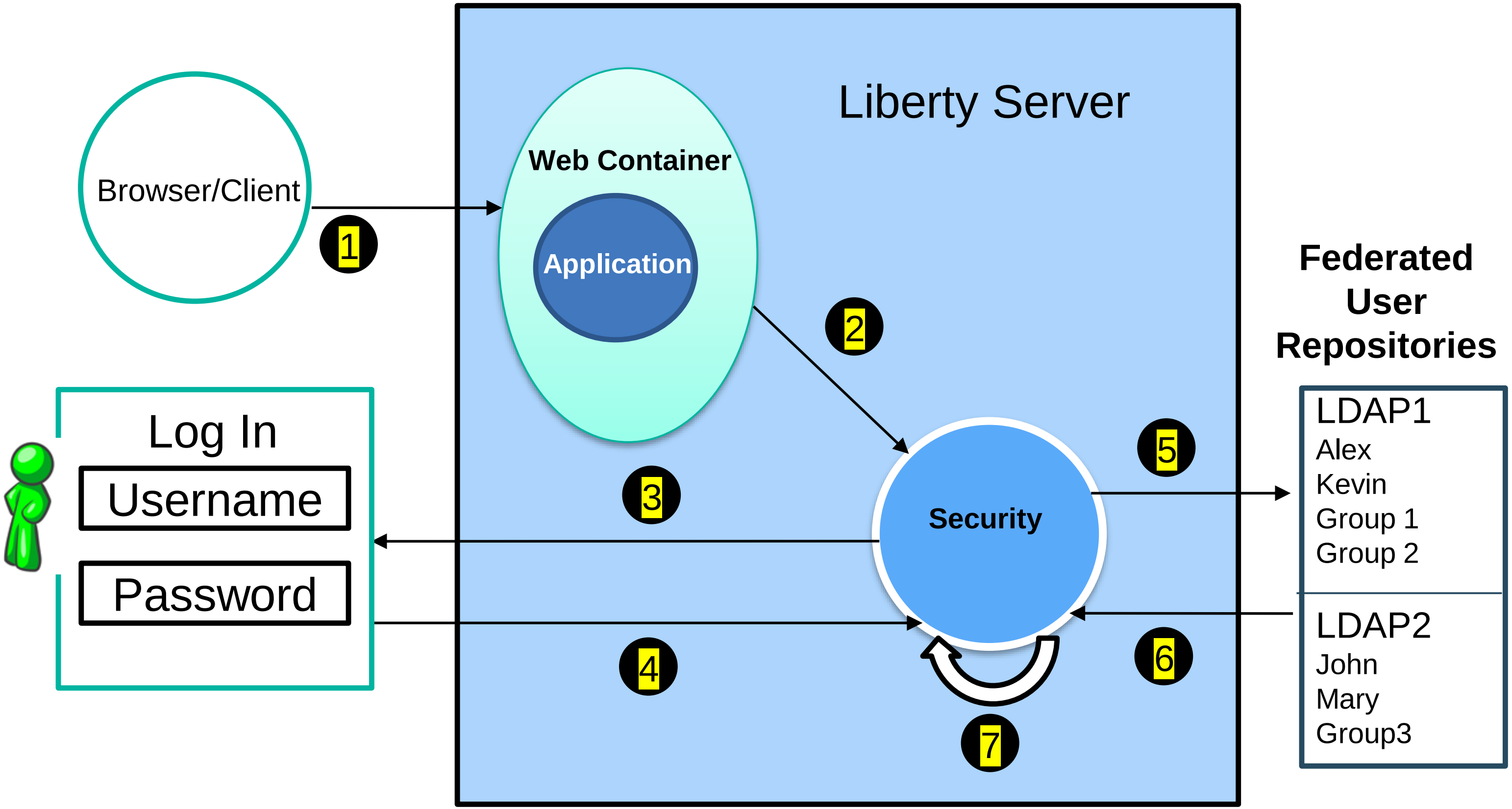
```
<feature>appSecurity-2.0</feature>
<feature>zosSecurity-1.0</feature>

<safRegistry id="saf" realm="MYPLEX" />

<safCredentials unauthenticatedUser=”MYDFLTU” profilePrefix=”MDEF" />

<safAuthorization id="saf” racRouteLog="ASIS" enableDelegation="true" />

<safRoleMapper profilePattern="%profilePrefix%.%resource%.%role%” toUpperCase="false" />
```

What’s needed with SAF on z/OS?


Angel Process
1
Yes
2
SERVER
BBG.AUTHMOD.BBGZSAFM.SAFCRED
BBG.SECPFX.<profile_prefix>
No
READ?
Fail
6
APPL
<profile_prefix>
No
Liberty z/OS Server
READ?
Yes
server.xml
3
5
4
User attempting to authenticate
Unauthenticated User (the "Default" User)

1. Liberty z/OS Angel Process available to the server
2. SERVER SAF profiles with the server ID having READ
   - BBG.AUTHMOD.BBGZSAFM.SAFCRED with server ID = READ
   - BBG.SECPFX.<*profile_prefix*> where the prefix value is related to your server prefix
   - Server ID granted READ to this SECPFX profile
3. The server.xml specifies SAF and names prefix value
4. A defined "unauthenticated" (i.e. "default") user
   - This is the ID that is used prior to successful authentication
   - This ID should have no TSO segment, and be RESTRICTED
5. User authenticating must have valid SAF definition (OMVS segment, valid home directory, not revoked)
6. APPL profile with READ to required IDs
   - The APPL profile is equal to the <*profile_prefix*> value you defined on the SERVER profile (#2)
   - The server ID has READ to this APPL
   - The unauthenticated user has READ to this APPL
   - The ID attempting to authenticate has READ to this APPL

Federation of user repositories



1. Access protected resource
2. Web Container calls security runtime
3. Security runtime prompts user for credentials
4. User enters credentials
5. Security runtime checks credentials against the registry – which is a **federation of user repositories**
6. Registry returns if credentials are valid or not
7. If valid, proceed to authorization

Federated Repositories

- Federated Repositories are used to federate multiple repositories into a single logical registry.
  - Very useful when the user/group data is spread across multiple repositories
  - For example, when one company buys/integrates with other company, they would have different user repositories.
- Federation of multiple LDAP repositories is common with many advanced customers.
- This feature is enabled by the federatedRepository-1.0 feature.
- This feature also supports the use of additional attributes in the LDAP server that basic LDAP configuration does not support.

Federated Repositories server.xml example

- IBM Directory Server

```
<ldapRegistry id="TDS” realm="SampleLdapIDSRealm" host=“myHost1" port=“9443" ignoreCase="true“
baseDN="o=ibm,c=us“ …….
```

- Active Directory

```
<ldapRegistry id="AD” realm="SampleLdapADRealm" host=“myHost2" port=“9443" ignoreCase="true"
baseDN="cn=users,dc=austin,dc=com“ …….
```

- Federation

```
<federatedRepository>
  <primaryRealm name="FederationRealm">
     <participatingBaseEntry name="o=ibm,c=us"/>
     <participatingBaseEntry name="cn=users,dc=austin,dc=ibm,dc=com"/>
   </primaryRealm>
</federatedRepository>
```

More on user registries …

- In addition to UserRegistry being an SPI, there is also an API
  - This implies that applications can call the UR methods to get user and group information just like the security service.
  - Applications can perform their own customized checking of user credentials and get the user/group information for performing additional checks.
  - Refer to : com.ibm.wsspi.security.registry.RegistryHelper
    - For more search on rwlp_sec_apis__example01

- System for Cross-domain Identity Management (SCIM)
  - Offers a standard set of APIs defined by the SCIM 1.1 specification to handle user and group information.
  - Liberty profile supports SCIM using the scim-1.0 feature.
  - Provides retrieval of user and group information using any attributes that are defined in the LDAP registry.
  - Also need the federatedRespository-1.0 feature to support additional attributes from LDAP.

More on user registries …

- Custom Adapter
  - In addition to Custom user Registry (CUR), a custom adapter SPI is also supported.
  - The custom adapter SPI allows both read and write capabilities
    - CUR only provides read capability
  - SPI is com.ibm.wsspi.security.wim.CustomRepository
  - The SPI uses the com.ibm.wsspi.security.wim.modelRoot.Root object to handle the UR attributes.
  - It is implemented as a user extension feature.

Authentication

Web application security

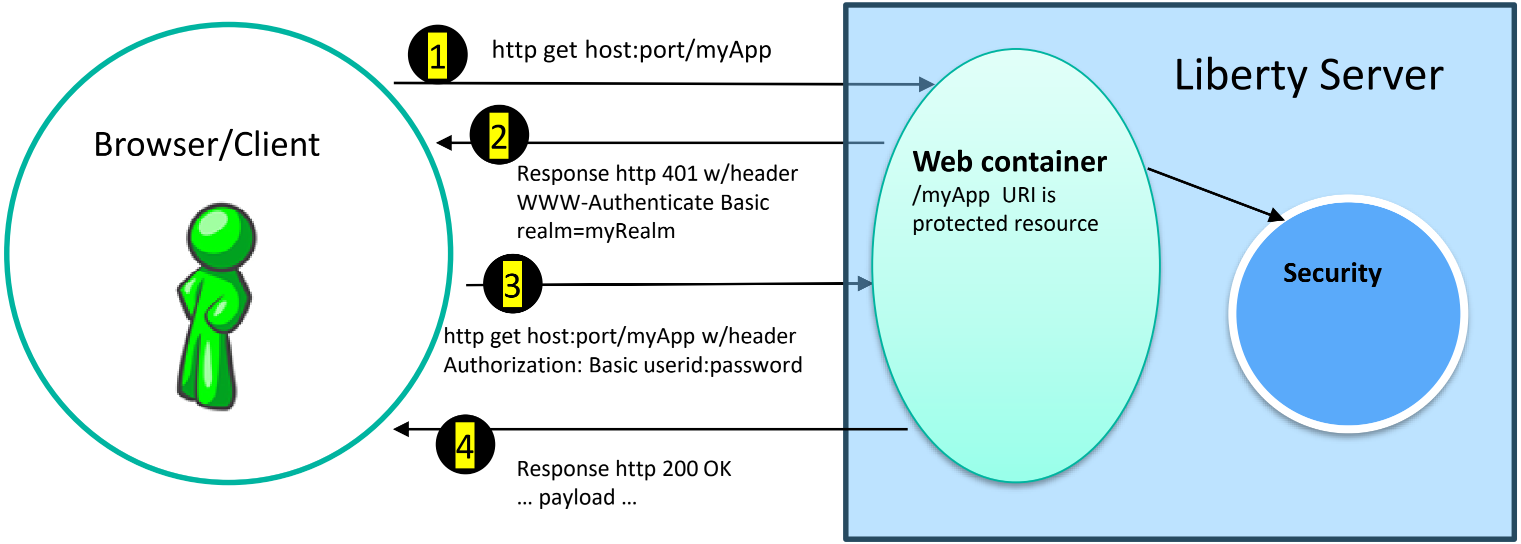
March 2018

Review of web application authentication

- With Java EE and web applications, there are several ways that authentication is done.
    1. Basic authentication
    2. Form based authentication
    3. Client authentication
    4. Mutual authentication

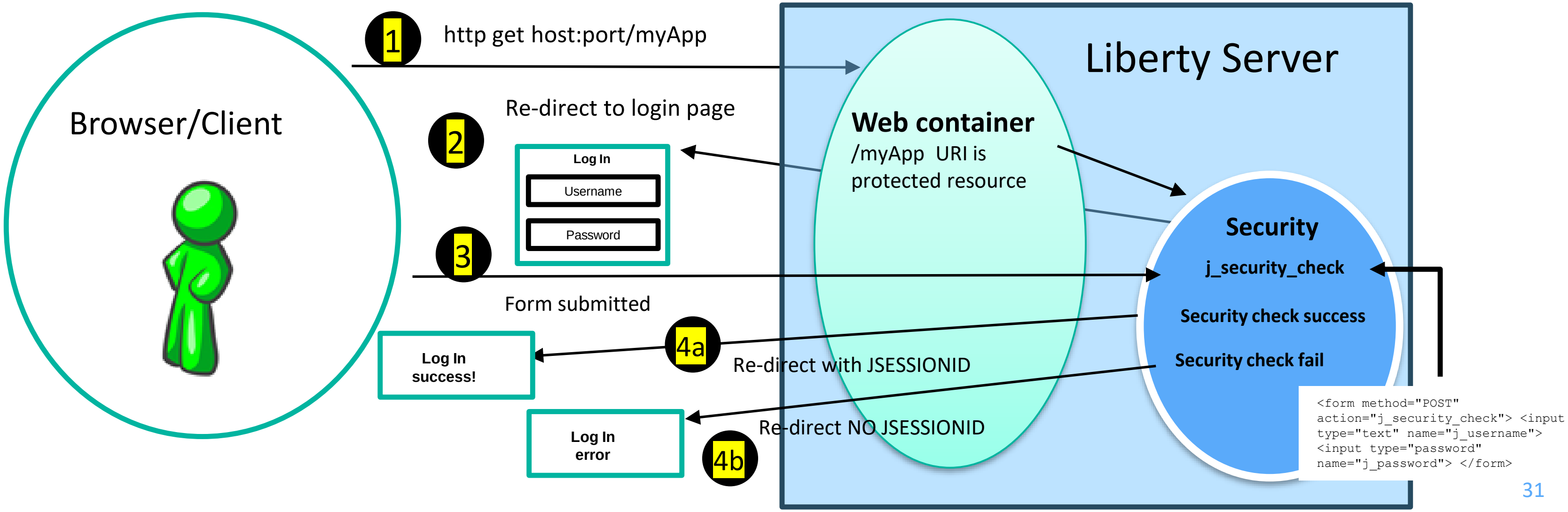
Basic authentication

1. Client requests access to a ‘protected resource’.
2. Server responds with an authentication challenge (HTTP 401 WWW-Authenticate and the realm name).
3. Client re-submits request adding Authorization: Basic userid:password (64bit encoded).
4. Server authenticates with user for the specified realm. If successful, returns HTTP 200 and result of request. If not, again returns HTTP 401 Unauthorized with WWW-Authenticate and the realm name,



Form-based authentication

1. Client requests access to a ‘protected resource’.
2. If client is unauthenticated, server re-directs client to login page.
3. Client submits login form to the server with username and password.
4. Server attempts to authenticate user.
   a) If authentication succeeds, user’s principal is checked to ensure it has the role that is needed to access the resource. If user is authorized, the server re-directs the client to the resource using stored URL path. Server passes back JESSIONID cookie with session context that is used for subsequent calls.
   b) If authentication fails, or the user is not in the proper role, client forwarded or re-directed to an error page.



Browser/Client

1 http get host:port/myApp

2 Re-direct to login page

Log In

Username

Password

3 Form submitted

4a Re-direct with JSESSIONID

Log In success!

4b Re-direct NO JSESSIONID

Log In error

Liberty Server

Web container

/myApp URI is protected resource

Security

j_security_check

Security check success

Security check fail

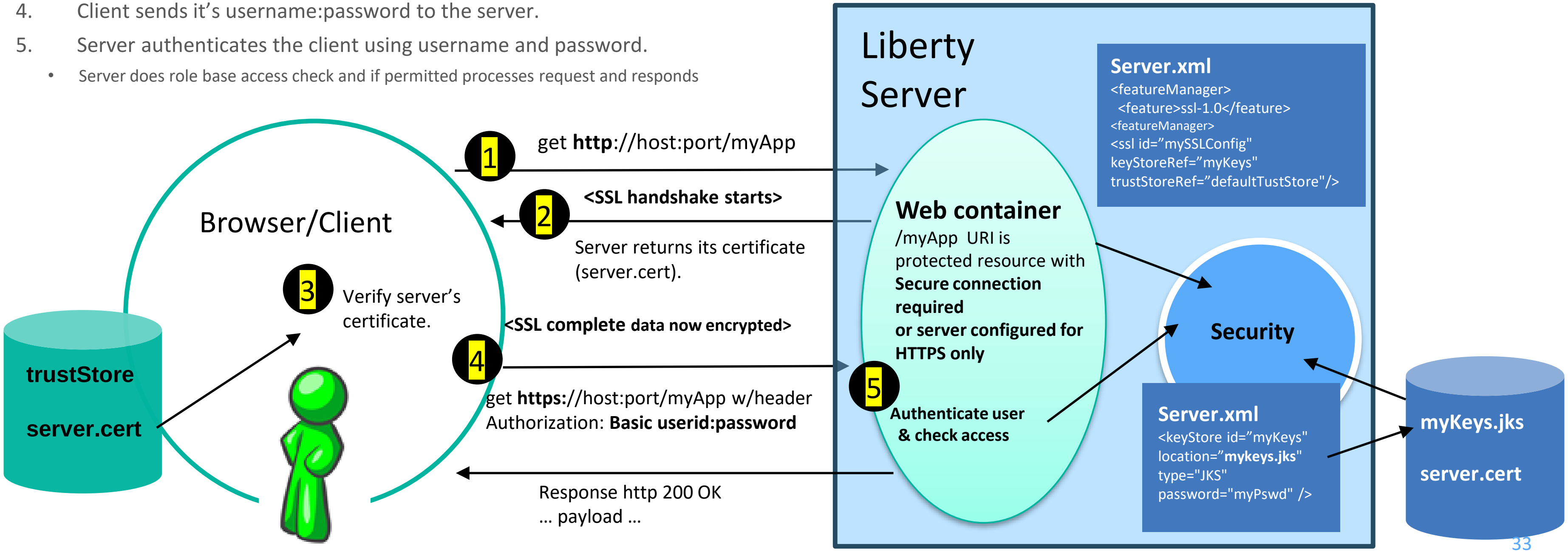
```
<form method="POST"
action="j_security_check"> <input
type="text" name="j_username">
<input type="password"
name="j_password"> </form>
```

Client certificate authentication

- With *client authentication*, the server authenticates the client using the client's public key certificate.
- Client authentication runs over a secured HTTP session over SSL/TLS (HTTPS).
- SSL/TLS is used to provide data encryption, server authentication, message integrity, and optional client authentication for a TCP/IP connection.
  - Public key certificates are the digital equivalent of a passport. The certificate is issued by a trusted organization, a certificate authority (CA), and provides identification for the bearer.

User name/password over SSL (aka 1-way SSL)

1. Client requests access to a ‘protected resource’.
   - SSL handshake begins with Client Hello message and server responds with Server Hello (each agreeing on cipher suite -server has control over what is selected)
2. Server presents its certificate to the client (signed by a certificate authority/CA, or self-signed, and containing it’s public key).
3. Client verifies server’s certificate (using it’s store of CA public keys to ensure the server is who it claims to be).
   - If server certificate (common name, date, issuer) is verified, client creates ‘pre-master’ secret (for RSA) for session, encrypts this with server’s public key and sends it to the server.
   - Server uses it’s private key to decrypt the pre-master secret. Both server and client negotiate the value of a ‘master key’ which is never actually sent over the wire.
   - **Session is NOW ENCRYPTED**
4. Client sends it’s username:password to the server.
5. Server authenticates the client using username and password.
   - Server does role base access check and if permitted processes request and responds



Browser/Client

1 get **http**://host:port/myApp

2 **<SSL handshake starts>**

Server returns its certificate (server.cert).

3 Verify server’s certificate.

trustStore

server.cert

<SSL complete data now encrypted>

4 get **https:**//host:port/myApp w/header
Authorization: **Basic userid:password**

Response http 200 OK
… payload …

Liberty Server

Web container
/myApp URI is protected with **Secure connection required or server configured for HTTPS only**

5 **Authenticate user & check access**

Security

Server.xml

```
<featureManager>
  <feature>ssl-1.0</feature>
<featureManager>
<ssl id=”mySSLConfig"
keyStoreRef=”myKeys"
trustStoreRef=”defaultTustStore"/>
```

Server.xml

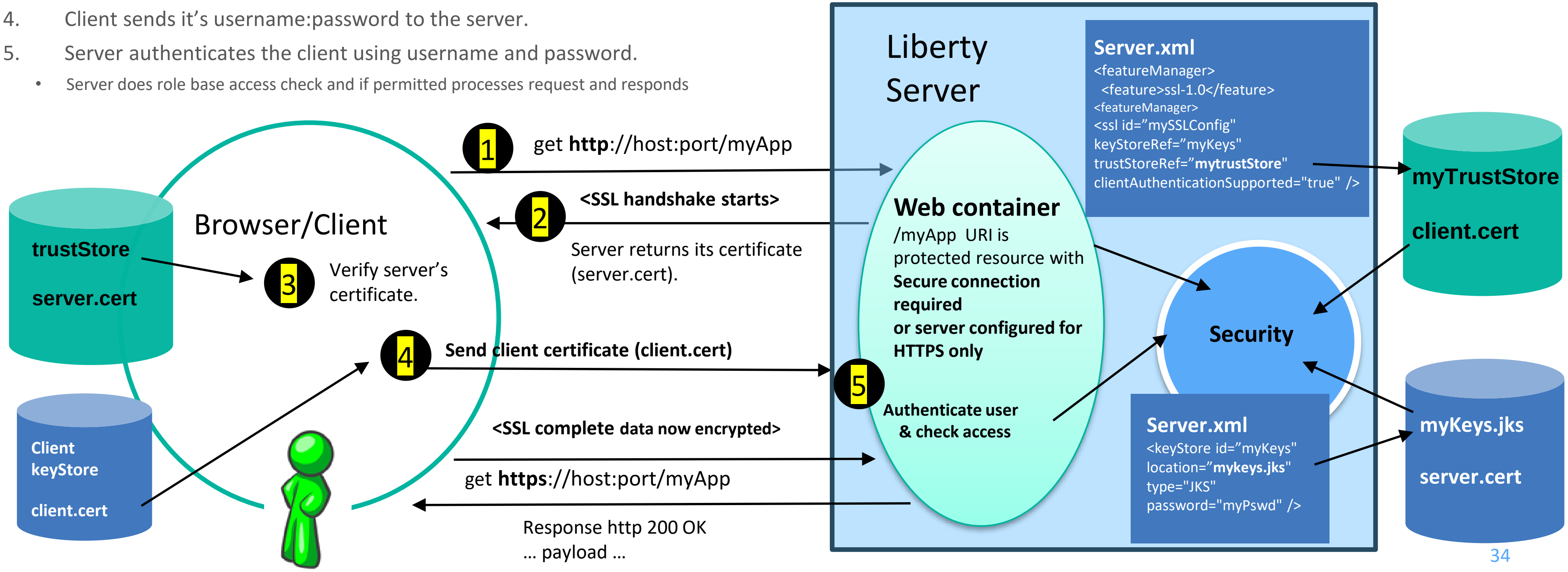
```
<keyStore id=”myKeys"
location=”mykeys.jks"
type="JKS"
password="myPswd" />
```

myKeys.jks

server.cert

Certificate based mutual authentication (aka 2-way SSL)

1. Client requests access to a ‘protected resource’.
   - SSL handshake begins with Client Hello message and server responds with Server Hello (each agreeing on cipher suite -server has control over what is selected)
2. Server presents its certificate to the client (signed by a certificate authority/CA, or self-signed, and containing it’s public key).
3. Client verifies server’s certificate (using it’s store of CA public keys to ensure the server is who it claims to be).
   - If server certificate (common name, date, issuer) is verified, client creates ‘pre-master’ secret (for RSA) for session, encrypts this with server’s public key and sends it to the server.
   - Server uses it’s private key to decrypt the pre-master secret. Both server and client negotiate the value of a ‘master key’ which is never actually sent over the wire.
   - **Session is NOW ENCRYPTED**
4. Client sends it’s username:password to the server.
5. Server authenticates the client using username and password.
   - Server does role base access check and if permitted processes request and responds



Liberty Server

Server.xml

```
<featureManager>
  <feature>ssl-1.0</feature>
<featureManager>
<ssl id=”mySSLConfig"
keyStoreRef=”myKeys"
trustStoreRef="mytrustStore"
clientAuthenticationSupported="true" />
```

myTrustStore

client.cert

trustStore

server.cert

Browser/Client

1 get **http**://host:port/myApp

2 **<SSL handshake starts>**

Server returns its certificate (server.cert).

3 Verify server’s certificate.

Web container

/myApp URI is protected resource with **Secure connection required or server configured for HTTPS only**

Security

4 **Send client certificate (client.cert)**

5 **Authenticate user & check access**

Client keyStore

client.cert

<SSL complete data now encrypted>

get **https**://host:port/myApp

Response http 200 OK
… payload …

Server.xml

```
<keyStore id=”myKeys"
location=”mykeys.jks"
type="JKS"
password="myPswd" />
```

myKeys.jks

server.cert

Certificate generation in Liberty

- There are several ways to generate certificates for a Liberty server
  1. securityUtility command line interface (CLI)
     - cd …/wlp/bin directory
     - securityUtility createSSLCertificate --server=*server_name* --password=*your_password*

     https://www.ibm.com/support/knowledgecenter/en/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/rwlp_command_securityutil.html
  2. WebSphere Developer Tools (WDT) –eclipse based tooling
     - Navigate to create certificate page
       - Utilities->Create SSL certificate
         - In the Keystore password field, type a password for your SSL certificate.
         - Click the Specify validity period (days) field, specify the number of days you want certificate to be valid for. Minimum length of time is 365 days.
         - Click the Specify subject (DN): field and provide a value for your SSL subject.
         - Click finish

     https://www.ibm.com/support/knowledgecenter/en/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/t_createsslcertificate.html

SSL Keystores and Certificates in Liberty

- Default keystore created for a Liberty server when SSL is enabled:

server.xml

```
<featureManager>
  <feature>ssl-1.0</feature> or
  <feature>transportSecurity-1.0</feature>
<featureManager>

<keyStore id=“defaultKeyStore"
    password=“yourPassword”  />
```

Default location of key store :

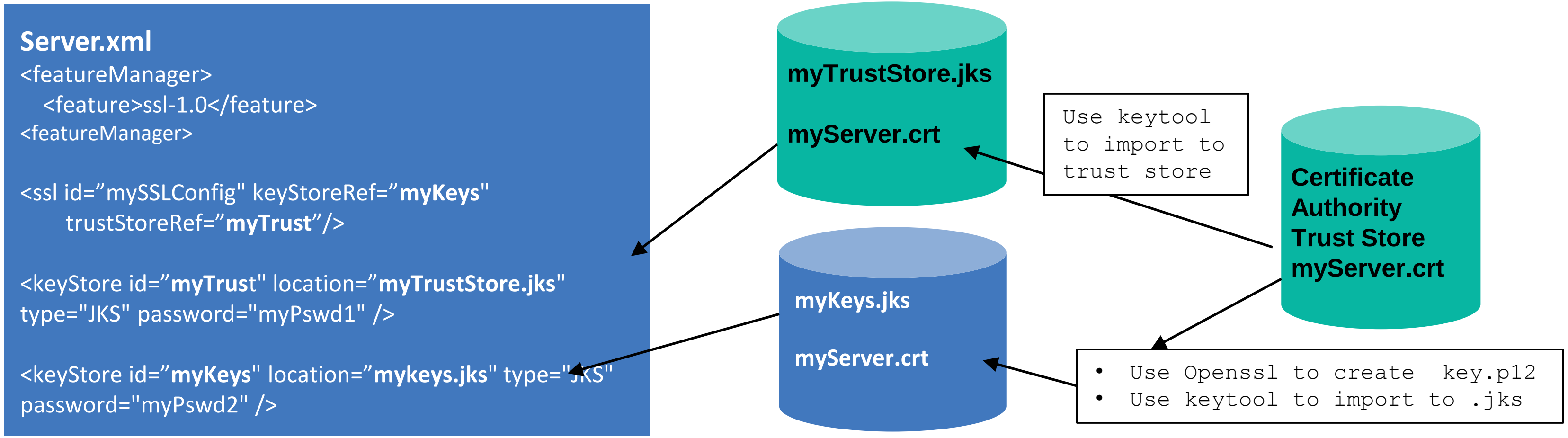
`/<WLP_USER_DIR>/myServer/resources/security/key.jks`

- When server starts …
  - Default keystore location: the file is called **key.jks** and is in the server or clients resources/security directory.
  - Keystore type: The **keystore type is JKS.**
  - Password: **comes from the configuration**.
  - Default certificate that is created by Liberty: Type: The **certificate is a self-signed certificate.**
  - Size: The default certificate size is 2048.
  - Signature algorithm: The **signature algorithm for the certificate is SHA256WITHRSA.**
  - Validity: The certificate is **valid for 365 days.**
  - SubjectDN: The certificate gets created with: **CN=<hostname>,OU=<client or server name>, O=ibm,C=US as the SubjectDN.**

https://www.ibm.com/support/knowledgecenter/en/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/rwlp_liberty_ssl_defaults.html

What about CA and 3rd party issued certificates?

- For production purposes, self signed certificates are not likely to be used. So how do I configure Liberty with Certificate Authority certs?
  - Apply for signed CA cert from Certificate Authority and put it in trustStore. An example showing the process:
    1. Export the server CA certificate and key in Public Key Cryptography standards 12 format.
       openssl pkcs12 -export -in myServer.crt -inkey server.key -out key.p12 -name default -passout pass:mypassword
    2. Import the server PKCS12 file to the server’s keystore.jks file for the server (below shown as myKeys.jks).
       keytool -importkeystore -deststorepass mypassword -destkeypass mypassword -destkeystore myKeys.jks -srckeystore key.p12 -srcstoretype PKCS12 -srcstorepass mypassword -alias default
    3. Import the server CA certificate to the keystore trust.jks file for the server (below shown as myTrustStore.jks).
       keytool -importcert -keystore myTrustStore.jks -storepass mypassword -file myServer.crt -alias default -noprompt



Server.xml

```
<featureManager>
   <feature>ssl-1.0</feature>
<featureManager>

<ssl id=”mySSLConfig" keyStoreRef=”myKeys"
      trustStoreRef=”myTrust”/>

<keyStore id=”myTrust" location=”myTrustStore.jks"
type="JKS" password="myPswd1" />

<keyStore id=”myKeys" location=”mykeys.jks" type="JKS"
password="myPswd2" />
```

myTrustStore.jks

myServer.crt

Use keytool to import to trust store

Certificate Authority Trust Store myServer.crt

myKeys.jks

myServer.crt

- Use Openssl to create key.p12
- Use keytool to import to .jks

https://www.ibm.com/support/knowledgecenter/en/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_icp_auto_ssl3.html

What about certificates and z/OS SAF keyrings?

- How do I configure z/OS SAF keyrings for my certificates?

server.xml

```
<featureManager>
  <feature>ssl-1.0</feature> or
  <feature>transportSecurity-1.0</feature>
  <feature>zosSecurity-1.0</feature>
<featureManager>

<keyStore id="defaultKeyStore"
      location="safkeyring:///WASKeyring"
      type="JCERACFKS" password="password"
      fileBased="false" readOnly="true" />
```

The RACF® key ring needs to be set up before you configure them for use by the Liberty server. The server does not create certificates and add them to RACF.

Sample RACF commands:

```
RACDCERT CERTAUTH GENCERT SUBJECTSDN(CN('CA for Liberty’)
           OU('LIBERTY’)) WITHLABEL('LibertyCA.LIBERTY') TRUST
           NOTAFTER(DATE(2018/12/31))
RACDCERT ADDRING(Keyring.LIBERTY) ID(aaaaaa)
RACDCERT ID (aaaaaa) GENCERT SUBJECTSDN(CN('yourhost.com’)
           O('IBM') OU('LIBERTY'))WITHLABEL('DefaultCert.LIBERTY’)
           SIGNWITH(CERTAUTH LABEL('LibertyCA.LIBERTY’))
           NOTAFTER(DATE(2018/12/31))
RACDCERT ID(aaaaaa) CONNECT (LABEL('DefaultCert.LIBERTY’)
           RING(Keyring.LIBERTY) DEFAULT)
RACDCERT ID(aaaaaa) CONNECT (RING(Keyring.LIBERTY)
           LABEL('LibertyCA.LIBERTY') CERTAUTH)
SETROPTS RACLIST(FACILITY) REFRESH
```

For details, refer to Liberty z/OS quick start guide here:
http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP102110

Single sign-on

March 2018

Single sign-on support in Liberty

- We'll review and compare a few options for single sign-on with Liberty. These include:
  1. LTPA
  2. SAML
  3. SPNEGO
  4. OAuth 2
  5. OpenID Connect (Server and Client)
     - JSON web tokens (JWTs) as a core component
     - Social Login

Liberty & Single sign-on – let’s start with LTPA

- Light weight third-party authentication (LTPA)
  - LTPA uses a shared set of keys between servers to enable cryptographic signatures in tokens that represent user sessions.
  - LTPA returns a token in an HTTP cookie, called **ltpaToken2** (can be customized)
  - LTPA tokens are time sensitive. Default expiration time is 120 mins. Servers sharing LTPA keys/tokens must have their time/date synchronized (coordinated UTC).
  - When appSecurity-2.0 is enabled in Liberty, LTPA keys are automatically created if they do not already exist.

server.xml

```
<featureManager>
  <feature>appSecurity-1.0</feature>
<featureManager>

<ltpa keysFileName=
     ”myLTPAKeysFileName.keys"
       keysPassword="keysPassword"
       expiration="120”
       monitorInterval="5s" />
```

Use securityUtility to encode or encrypt the keysPassword value.

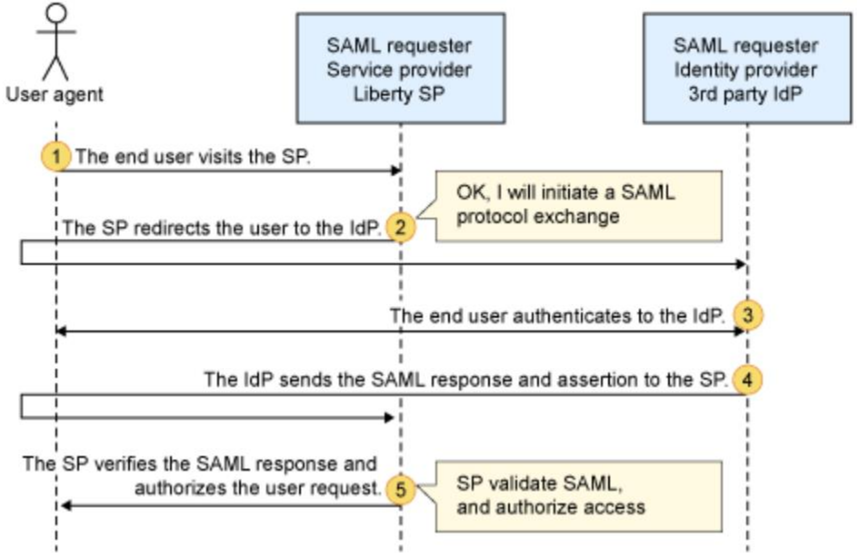
***NEW in 17.0.0.4: AES encryption for passwords (new wlp.password.encryption.key property)**

Default location of LTPA keys :

```
/<WLP_USER_DIR>/myServer/resources/security/ltpa.keys or
${server.output.dir}/resources/security/ltpa.keys
```

Liberty & Single sign-on – SAML 2.0

- Security Assertion Markup Language (SAML 2.0)
  - SAML is an Oasis standard for representing and exchanging user identity.
  - Web users authenticate to a SAML identity provider (IdP) and receive a SAML assertion.
  - 3 entities, 1) end user, 2) identity provider (IdP), and 3) service provider.
  - User always authenticates through IdP and SP uses IdP with assertion doc to identify the user.
  - Liberty feature : <feature>samlWeb-2.0</feature>

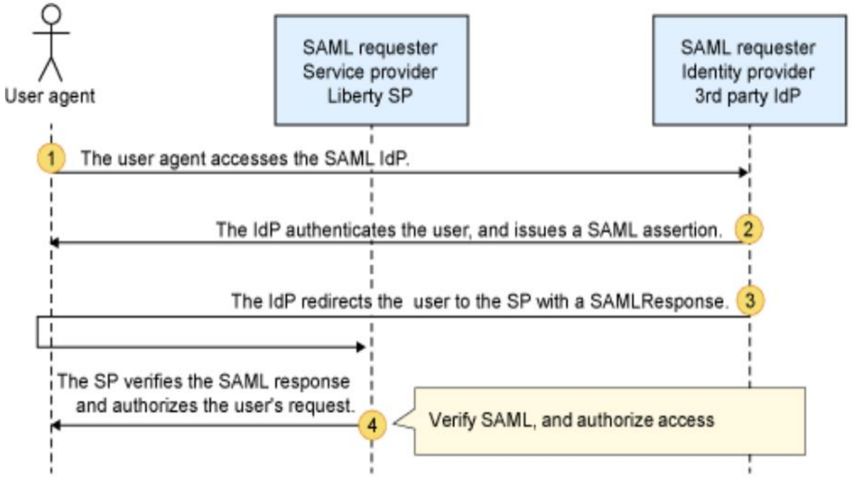

User agent
SAML requester
Service provider
Liberty SP
SAML requester
Identity provider
3rd party IdP
1 The end user visits the SP.
OK, I will initiate a SAML protocol exchange
The SP redirects the user to the IdP. 2
The end user authenticates to the IdP. 3
The IdP sends the SAML response and assertion to the SP. 4
The SP verifies the SAML response and authorizes the user request. 5
SP validate SAML, and authorize access

Service provider-initiated Web single sign-on (end user starting at SP)

1. End user visits the SP.
2. SP redirects the user to the IdP.
3. The end user authenticates to the IdP.
4. The IdP sends the SAML response and assertion to the SP.
5. The SP verifies the SAML response and authorizes the user request.

(because SAML token is standalone, no longer need access to user registry to check access)

Liberty & Single sign-on – SAML 2.0 (part 2)


User agent
SAML requester
Service provider
Liberty SP
SAML requester
Identity provider
3rd party IdP
1
The user agent accesses the SAML IdP.
The IdP authenticates the user, and issues a SAML assertion.
2
The IdP redirects the user to the SP with a SAMLResponse.
3
The SP verifies the SAML response
and authorizes the user's request.
4
Verify SAML, and authorize access

Identity provider-initiated Web single sign-on (end user starting at IdP)

1. End user visits the SAML IdP.
2. IdP authenticates user and issues a SAML assertion.
3. The IdP re-directs user to the SP with SAMLResponse.
4. The SP verifies the SAML response and authorizes the user request.

(used extensively by web portals)

https://www.ibm.com/support/knowledgecenter/en/SSEQTP_8.5.5/com.ibm.websphere.wlp.doc/ae/cwlp_saml_web_sso.html

Liberty & Single sign-on – SPNEGO

- Simple and Protected GSS-API Negotiation Mechanism (SPNEGO)
  - SPNEGO is a standard defined in IETF RFC 2478. When enabled in Liberty SPNEGO is initialized when the first HTTP request arrives.
  - Client applications – such as Microsoft .NET, web service and J2EE client that supports SPNEGO web applications.
    - SPNEGO is most commonly associated with Microsoft Active Directory
    - Users must request SPNEGO tokens from the Kerberos KDC and send the token to the server
    - SPNEGO web authentication code in the server validates the token and makes authorization decisions from this.
  - Server generates LTPA token and returns it in the HTTP response.

Features needed for SPNEGO in Liberty

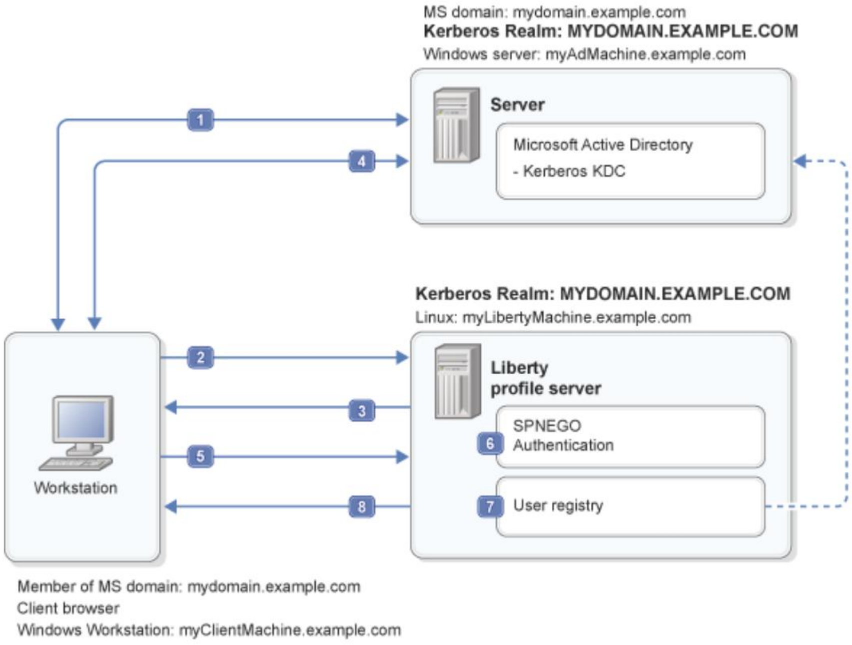
```
<featureManager>
    <feature>spnego-1.0</feature>
    <feature>appSecurity-2.0</feature> .
</featureManager>
```

Sample SNPEGO config

```
<spnego id="mySpnego"
    includeClientGSSCredentialInSubject="false"
    krb5Config=
"${server.config.dir}/resources/security/kerberos/k
rb5.conf"
    krb5Keytab=
"${server.config.dir}/resources/security/kerberos/k
rb5.keytab"
    servicePrincipalNames=
"HTTP/myLibertyMachine.example.com"
    authFilterRef="myAuthFilter" /> </spnego>
```

https://www.ibm.com/support/knowledgecenter/en/SS7K4U_liberty/com.ibm.websphere.wlp.zseries.doc/ae/twlp_spnego_config.html

Liberty & Single sign-on – SPNEGO

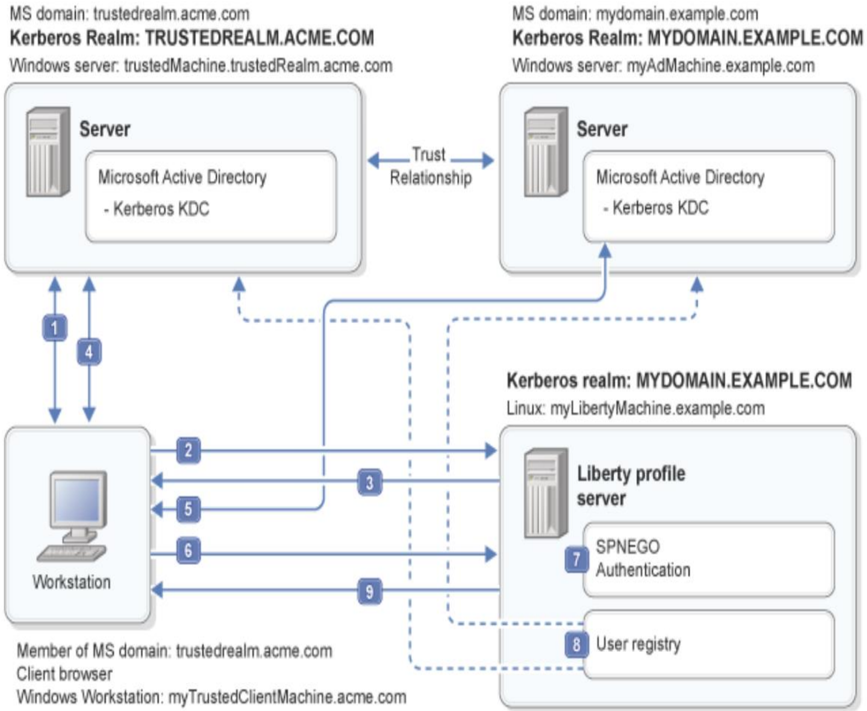

MS domain: mydomain.example.com
Kerberos Realm: MYDOMAIN.EXAMPLE.COM
Windows server: myAdMachine.example.com
Server
Microsoft Active Directory
- Kerberos KDC
1
4
Kerberos Realm: MYDOMAIN.EXAMPLE.COM
Linux: myLibertyMachine.example.com
Liberty
profile server
SPNEGO
Authentication
User registry
2
3
5
6
7
8
Workstation
Member of MS domain: mydomain.example.com
Client browser
Windows Workstation: myClientMachine.example.com

SPNEGO web authentication in a single Kerberos realm

1. User logs on to Microsoft domain controller (MYDOMAIN.EXAMPLE.COM).
2. User requests protected web resource (HTTP GET).
3. SPNEGO support in Liberty responds with HTTP 401 unauthorized (with `Authenticate Negotiate` status).
4. Client/browser sees Negotiate and handshakes with KDC to get a Kerberos service ticket.
5. Client responds the server Negotiate with the Kerberos ticket (token) in the HTTP request header.
6. Liberty retrieves the SPNEGO token, validates it and retrieves the identity/principal for the user.
7. Liberty completes the authentication for the user from it’s registry and performs authorization checks.
8. If access permitted, Liberty executes the request and sends client HTTP 200 OK and any response payload.

https://www.ibm.com/support/knowledgecenter/en/SSEQTP_8.5.5/com.ibm.websphere.wlp.doc/ae/cwlp_spnego.html

Liberty & Single sign-on – SPNEGO part 2


MS domain: trustedrealm.acme.com
Kerberos Realm: TRUSTEDREALM.ACME.COM
Windows server: trustedMachine.trustedRealm.acme.com
Server
Microsoft Active Directory
- Kerberos KDC
Trust Relationship
MS domain: mydomain.example.com
Kerberos Realm: MYDOMAIN.EXAMPLE.COM
Windows server: myAdMachine.example.com
Server
Microsoft Active Directory
- Kerberos KDC
1
4
Kerberos realm: MYDOMAIN.EXAMPLE.COM
Linux: myLibertyMachine.example.com
2
3
5
6
9
Workstation
Liberty profile server
7
SPNEGO Authentication
8
User registry
Member of MS domain: trustedrealm.acme.com
Client browser
Windows Workstation: myTrustedClientMachine.acme.com

https://www.ibm.com/support/knowledgecenter/en/SSEQTP_8.5.5/com.ibm.websphere.wlp.doc/ae/cwlp_spnego.html

SPNEGO web authentication in trusted Kerberos realms

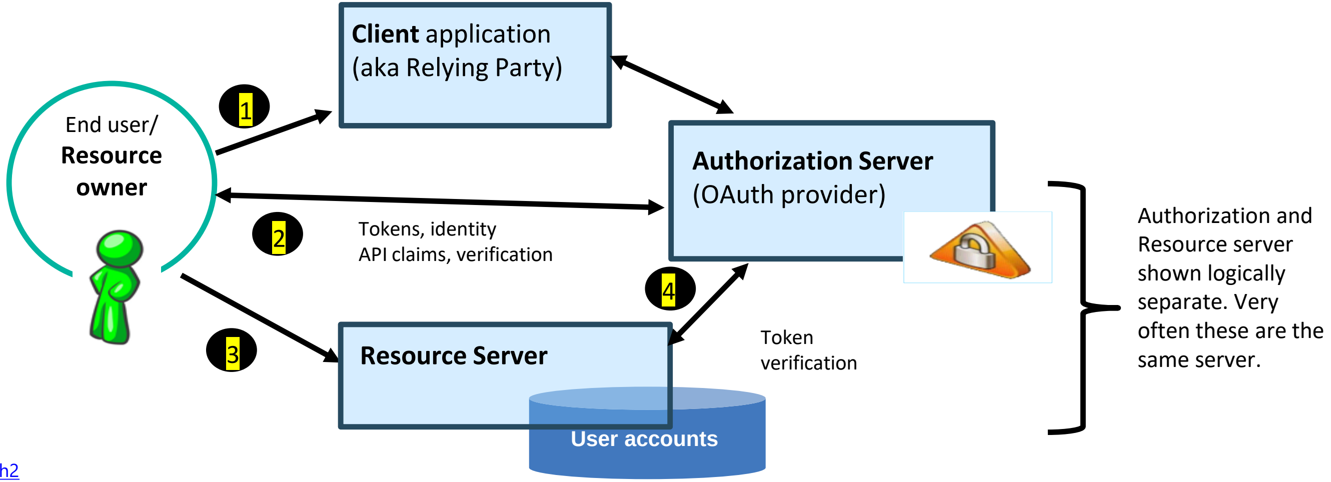
1. User logs on to Microsoft domain controller (TRUSTEDREALM.ACME.COM).
2. User requests protected web resource (HTTP GET) on MYDOMAIN.EXAMPLE.COM.
3. SPNEGO support in Liberty responds with HTTP 401 unauthorized (with `Authenticate Negotiate` status).
4. Client/browser sees Negotiate and handshakes with KDC on TRUSTEDREALM.ACME.COM – requesting MYDOMAIN.EXAMPLE.COM and gets a Kerberos cross realm service ticket.
5. Client/browser uses cross realm ticket to call KDC in MYDOMAIN.EXAMPLE.COM, retrieving another service ticket.
6. Client responds the server Negotiate with the Kerberos service ticket (token) in the HTTP request header.
7. Liberty retrieves the SPNEGO token, validates it and retrieves the identity/principal for the user.
8. Liberty completes the authentication for the user from it’s registry and performs authorization checks.
9. If access permitted, Liberty executes the request and sends client HTTP 200 OK and any response payload.

Liberty & Single sign-on – OAuth 2.0

- OAuth 2.0 (spec finalized in 2012)
  - OAuth is a ‘delegation protocol’ for conveying authorization decisions across a network. It does NOT provide Authentication though it can be used as part of Authentication.
  - Key concepts –
    - Resource owner – grants access to protected resource – usually an end user
    - Resource server – server hosting protected resources – accepts/responds requests for resource using access tokens
    - Client – ‘application’ running anywhere that makes requests to Resource server on behalf of resource owner (aka ‘relying party’)
    - Authorization server – server that issues tokens to client after successfully driving authentication and obtaining user’s permissions

Features needed for OAuth 2.0 in Liberty

```
<featureManager>
    <feature>jsp-2.0</feature>
    <feature>servlet-3.0</feature>
    <feature>sss-1.0</feature>
    <feature>oauth-2.0</feature>
   <feature>appSecurity-1.0</feature> .
</featureManager>
```



Client application
(aka Relying Party)

End user/
Resource owner

1

Authorization Server
(OAuth provider)

2

Tokens, identity
API claims, verification

3

4

Resource Server

User accounts

Token
verification

Authorization and Resource server shown logically separate. Very often these are the same server.

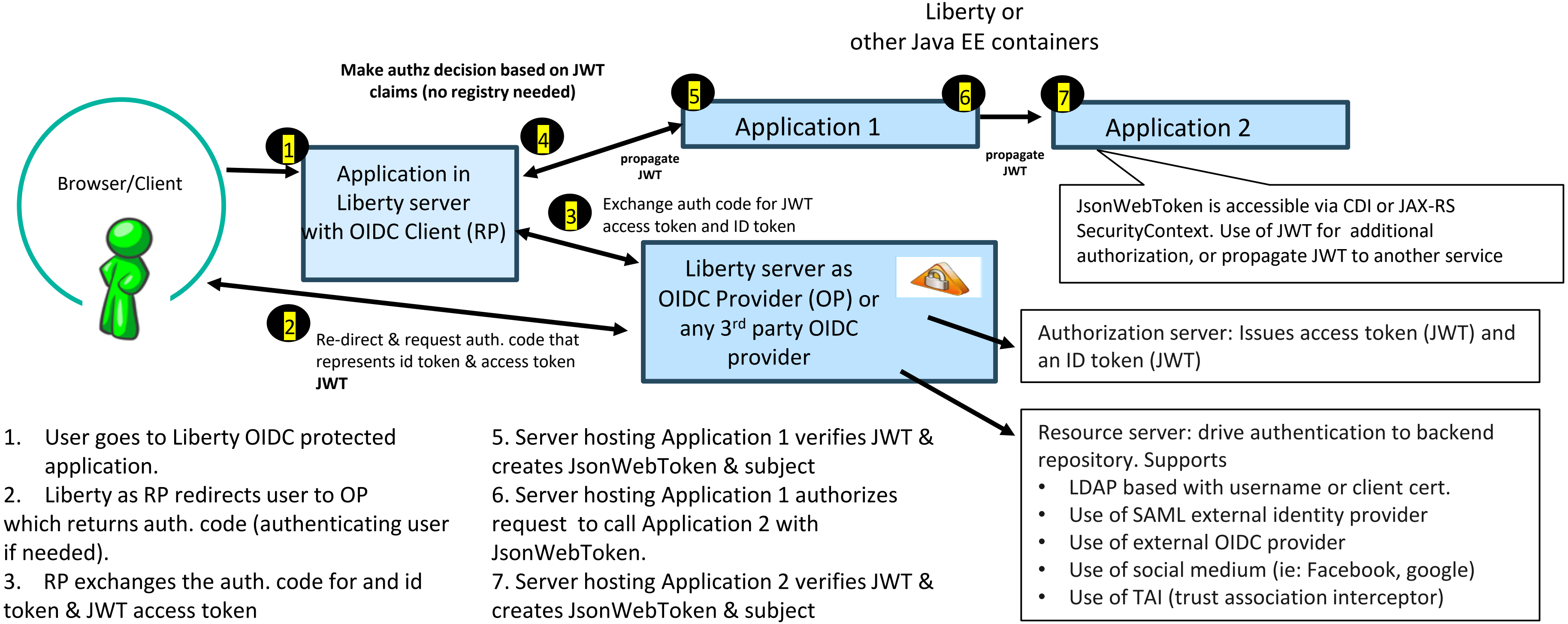
https://developers.google.com/identity/protocols/OAuth2
https://www.ibm.com/support/knowledgecenter/en/SSZSXU_6.2.2.7/com.ibm.tivoli.fim.doc_6227/config/concept/OAuth20Workflow.html

Liberty & Single sign-on – Open ID Connect (OIDC)

- OpenID Connect 1.0 (OIDC) is a simple identity protocol over OAuth 2.0. OIDC is a popular Internet SSO protocol, and it works well with cloud, mobile, and native applications.
- OIDC lets a client application request the identity of the user as an ID token in a standardized, REST-like manner. In addition, the client application can use access tokens to access REST-like Services.
- OIDC refers to 'OpenID Connect Client' as 'Relying Party' (RP) and authorization server as 'OpenID Connect Provider' (OP).
  - RP depends on the OP to provide secure tokens that represent users and their permissions.
- It is common for an OIDC provider to issue JSON Web tokens (JWTs) as access tokens.
  - JWTs provide a standardized way to represent information that is secure (contains digital signature)
- Liberty supports both JWT and OpenID Connect for single-sign on. Here are high level summaries of JWT and OpenID Connect functions in Liberty.
  - Liberty can be a dedicated OpenID Connect server for single-sign on.
  - Liberty can be OpenID Connect client and relying party.
  - Liberty can accept JWTs as authentication tokens.
  - Liberty provides APIs to self-issue JWTs, and consume JWTs.

Liberty & Single sign-on – Open ID Connect (OIDC)

- Open ID Connect for single sign-on.



Liberty or
other Java EE containers

Make authz decision based on JWT claims (no registry needed)

Browser/Client

1

Application in Liberty server with OIDC Client (RP)

4

propagate JWT

5

Application 1

6

propagate JWT

7

Application 2

JsonWebToken is accessible via CDI or JAX-RS SecurityContext. Use of JWT for additional authorization, or propagate JWT to another service

3

Exchange auth code for JWT access token and ID token

Liberty server as OIDC Provider (OP) or any 3rd party OIDC provider

2

Re-direct & request auth. code that represents id token & access token **JWT**

Authorization server: Issues access token (JWT) and an ID token (JWT)

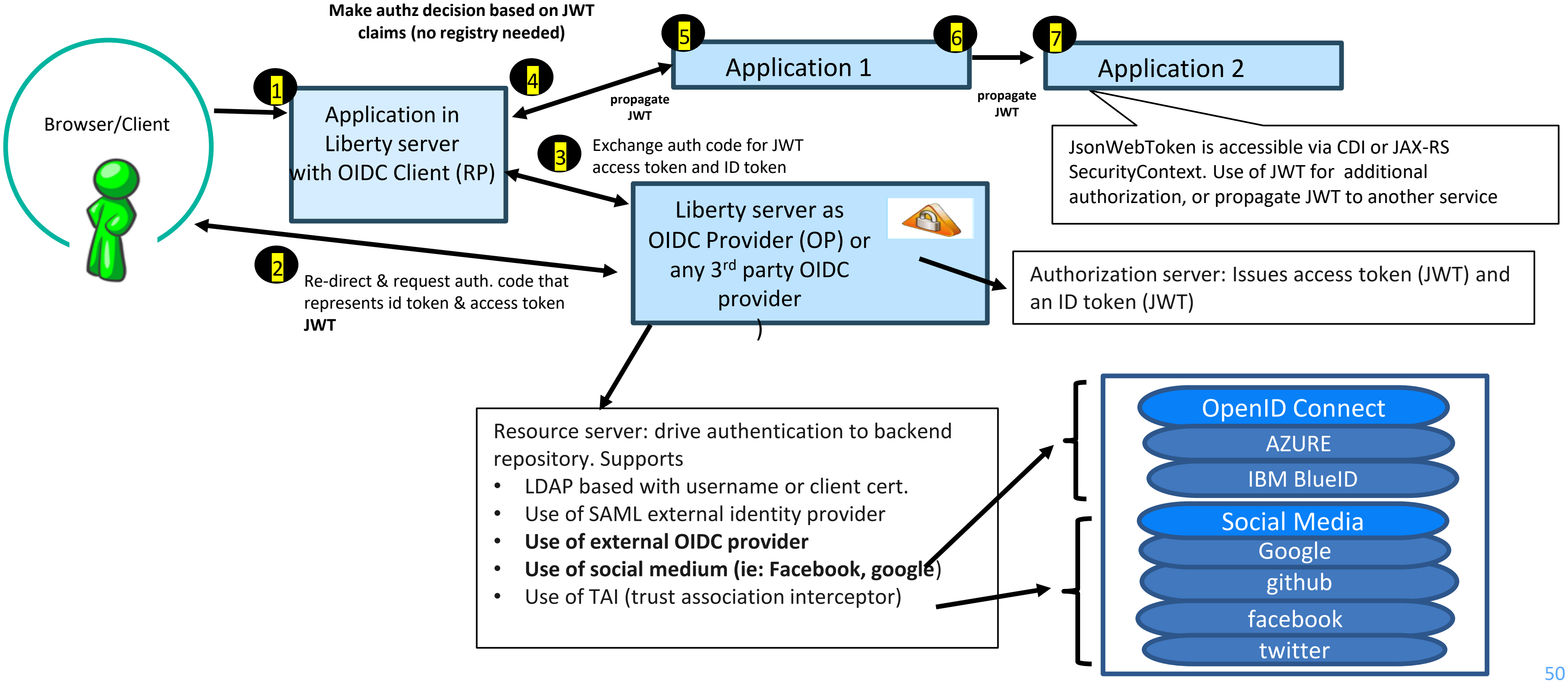
Resource server: drive authentication to backend repository. Supports

- LDAP based with username or client cert.
- Use of SAML external identity provider
- Use of external OIDC provider
- Use of social medium (ie: Facebook, google)
- Use of TAI (trust association interceptor)

1. User goes to Liberty OIDC protected application.
2. Liberty as RP redirects user to OP which returns auth. code (authenticating user if needed).
3. RP exchanges the auth. code for and id token & JWT access token
4. Application in Liberty RP makes call to Application 1 (JWT is propagated)
5. Server hosting Application 1 verifies JWT & creates JsonWebToken & subject
6. Server hosting Application 1 authorizes request to call Application 2 with JsonWebToken.
7. Server hosting Application 2 verifies JWT & creates JsonWebToken & subject

Liberty & Single sign-on – Social Media Login

- Social Login (feature SocialLogin-1.0)



Make authz decision based on JWT claims (no registry needed)

Browser/Client

1 Application in Liberty server with OIDC Client (RP)

2 Re-direct & request auth. code that represents id token & access token **JWT**

3 Exchange auth code for JWT access token and ID token

4 propagate JWT

5 Application 1 6

propagate JWT

7 Application 2

JsonWebToken is accessible via CDI or JAX-RS SecurityContext. Use of JWT for additional authorization, or propagate JWT to another service

Liberty server as OIDC Provider (OP) or any 3rd party OIDC provider

)

Authorization server: Issues access token (JWT) and an ID token (JWT)

Resource server: drive authentication to backend repository. Supports

- LDAP based with username or client cert.
- Use of SAML external identity provider
- **Use of external OIDC provider**
- **Use of social medium (ie: Facebook, google**)
- Use of TAI (trust association interceptor)

OpenID Connect
AZURE
IBM BlueID

Social Media
Google
github
facebook
twitter

Authorization

March 2018

JEE Role based authorization and Liberty

Who Are You?

This is ***authentication***. We looked at that earlier. Once a user successfully authenticates, the server knows who the user is.

What Are You Allowed To Do?

This is ***authorization***. It is a function of the application. An application may or may not define different "roles," but if roles are defined, then the server gets involved to help the application determine if a user is a member of the defined role.

Application WAR File web.xml

```
<servlet>
   <servlet-name>myHello</servlet-name>
   <servlet-class>HelloServlet</servlet-class>
   <security-role>
      <role-name>MyRole</role-name>
   </security-role>
</servlet>
```

role

Liberty server.xml

```
<feature>appSecurity-2.0</feature>

<application
   id="myServlet‶ name="myServletWAR" type="war"
   location="/<path>/myServletWAR.war" >
   <application-bnd>
      <security-role name="MyRole">
        <group name=”Operators" />
      </security-role>
   </application-bnd>
</application>
```

role

group

If the authenticated user is a member of this group, then they have access to that role.

If no application-bnd in server.xml (no explicit authorization) the default behavior is to use the group name for the role.

Other information

March 2018

Using securityUtility to encrypt passwords

- **securityUtility** –
  - Also supports an encode/encrypt for any string (AES) – such as for passwords
    - Where more protection is needed beyond encoding.
  - Use --key=encryption_key
    - Specifies the key to be used when encoding using AES encryption. This string is hashed to produce an encryption key that is used to encrypt and decrypt the password. The key can be provided to the server by defining the variable **wlp.password.encryption.key** whose value is the key. If this option is not provided, a default key is used.

https://www.ibm.com/support/knowledgecenter/en/SSEQTP_liberty/com.ibm.websphere.wlp.doc/ae/rwlp_command_securityutil.html

Few things we're working on now (looking for input)

- **Java EE 8 Security support (JSR 375)** –
  - Adds CDI annotations for much of the features that formerly required config. Such as :

```
 @WebServlet("/protectedServlet")
@ServletSecurity(@HttpConstraint(rolesAllowed = "foo"))
public class ProtectedServlet extends HttpServlet { ..
```

  - https://javaee.github.io/security-spec and https://jcp.org/en/jsr/detail?id=375
- **Auditing** – (now in beta)
- **JWT cookies** (as next generation of LTPA)

****** Reminder (AGAIN) Next Session ******

Security Hardening and Best Practices for Running WebSphere Liberty in Production –

April 6th at 11am ET - In this session, we'll cover some fundamental best practices related to Liberty and security in a production environment. http://ibm.biz/BdZEwB

Thank You

Your Feedback is Important!

March 2018