The Anatomy of a Secure Web App Using JavaEE, Spring Security and Apache Fortress

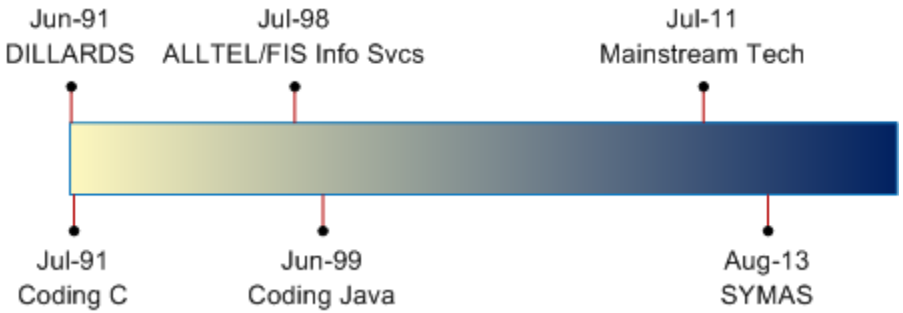
October 5, 2017

Little Rock Tech Fest

Objective

Think about how we should be securing web apps.

Introductions



Shawn McKinney

- symas Software Architect
- PMC Apache Directory Project
- OpenLDAP™ PROJECT Engineering Team

Agenda

Have a look at…

1. Java Remote Code Execution Vulnerability
   - https://github.com/shawnmckinney/serial-exploit-sample

2. Apache Fortress Demo + Java EE + Spring Security
   - https://github.com/shawnmckinney/apache-fortress-demo

3. Fortress SAML Demo + Spring Security SP
   - https://github.com/shawnmckinney/fortress-saml-demo

Recommendation

Listen and absorb *conceptually*. Slides will be published and have the *details*.

What's The Problem

- Equifax Breach
  - 143 million Americans' personal info, including names, addresses, dates of birth and SSNs compromised.
  - Only a veneer of security in place.

What's The Exploit

"The vulnerability was Apache Struts CVE-2017-5638 "

http://www.zdnet.com/article/equifax-confirms-apache-struts-flaw-it-failed-to-patch-was-to-blame-for-data-breach/

The Exploit

"The Jakarta Multipart parser in Apache Struts 2 2.3.x before 2.3.32 and 2.5.x before 2.5.10.1 mishandles file upload, which allows remote attackers to execute arbitrary commands via a #cmd= string in a crafted Content-Type HTTP header, as exploited in the wild in March 2017."

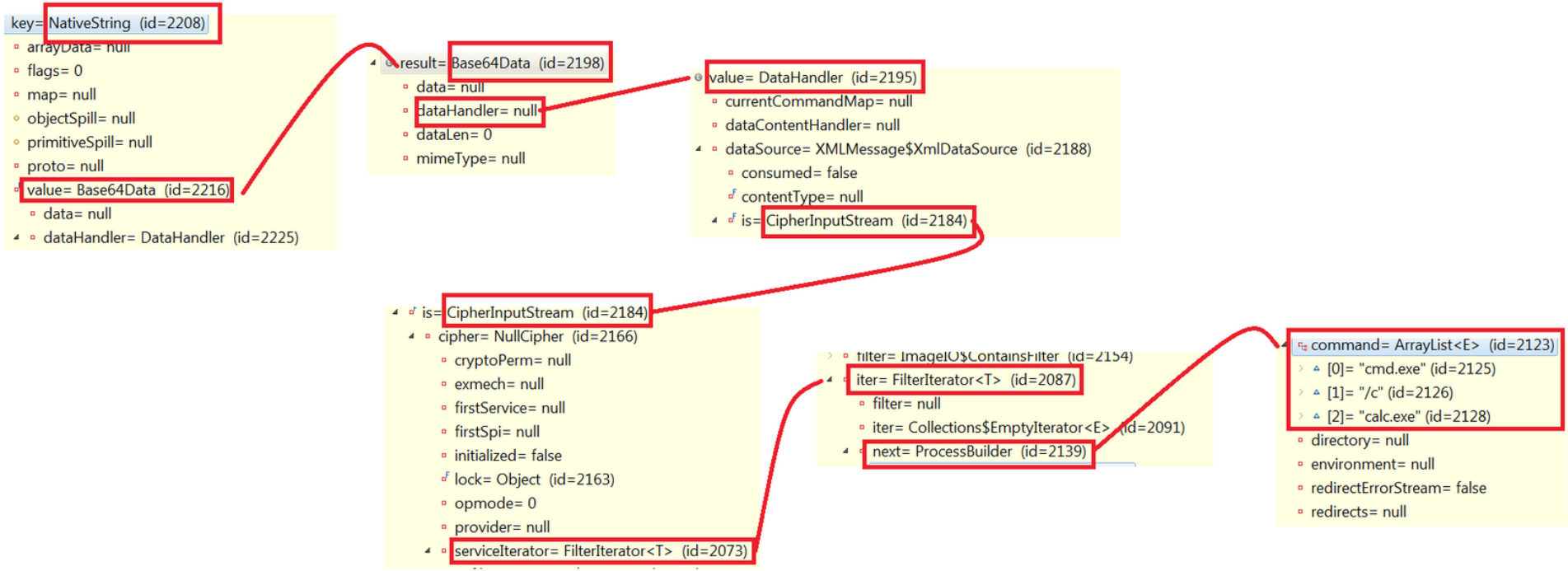
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5638

How it Works

- Input data deserialized into an executable object with privilege.

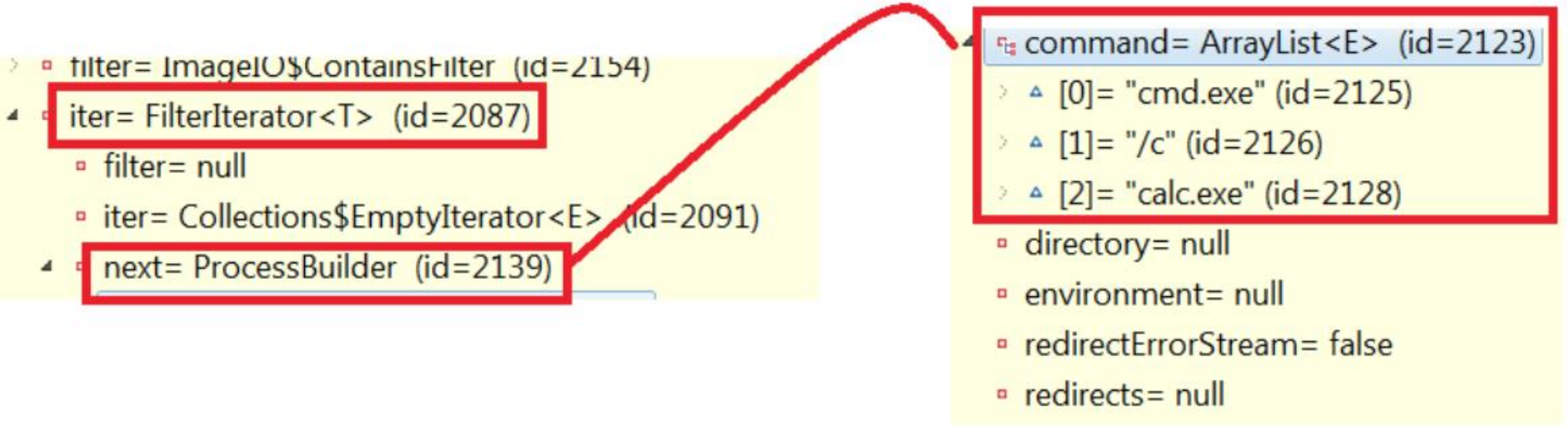
http://blog.diniscruz.com/2013/12/xstream-remote-code-execution-exploit.html

Apache Struts CVE-2017-9805



https://securingtomorrow.mcafee.com/mcafee-labs/apache-struts-at-rest-analyzing-remote-code-execution-vulnerability-cve-2017-9805/

Apache Struts CVE-2017-9805



https://securingtomorrow.mcafee.com/mcafee-labs/apache-struts-at-rest-analyzing-remote-code-execution-vulnerability-cve-2017-9805/

Equifax Breach

"Generally when you successfully exploit a web-application bug like this you will become the system user who owns the web server process,"

Alex McGeorge, the head of threat intelligence at the security firm Immunity.

https://www.wired.com/story/equifax-breach-no-excuse/

The Solution

Ensure all appropriate patches have been applied and that you aren't running software with vulnerabilities or backdoors.

How do we ensure that our software is free of vulnerabilities, known or otherwise?

How do we ensure that our software is free of vulnerabilities, known or otherwise?

can’t be done - practically

So Now What?

"Security best practices dictate that this user have as little privilege as possible on the server itself, since security vulnerabilities in web applications and web servers are so commonly exploited."

https://www.wired.com/story/equifax-breach-no-excuse/

The Solution (Take 2)

Practice the principle of least privilege.

Principle of least privilege

From Wikipedia, the free encyclopedia

https://en.wikipedia.org/wiki/Principle_of_least_privilege

Not to be confused with Rule of least power.

In information security, computer science, and other fields, the **principle of least privilege** (also known as the **principle of minimal privilege** or the **principle of least authority**) requires that in a particular abstraction layer of a computing environment, every module (such as a process, a user, or a program, depending on the subject) must be able to access only the information and resources that are necessary for its legitimate purpose.[1][2]

Java Object Serialization Exploit

```
public class BadCode
   implements java.io.Serializable…
{…
 private void
  readObject(java.io.ObjectInputStream in)
 {
    in.defaultReadObject();
    Runtime.getRuntime().exec( cmd );
```

Java's remote code execution exploit occurs when a rogue object is read from an input resource and deserialized.

Employ a Runtime Java Security Policy

```
grant codeBase "file:${catalina.home}/webapps/my-web-app-1/-" {
    permission java.net.SocketPermission "localhost", "resolve";
    permission java.net.SocketPermission "127.0.0.1:32768", "connect,resolve";
    permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
    permission java.io.SerializablePermission "enableSubclassImplementation";
    permission java.io.FilePermission "…/resources/", "execute";
    …
    };
```

^ use w/ caution

Demo # 1

https://github.com/shawnmckinney/serial-exploit-sample

Not a Perfect Solution

```
grant codeBase "file:${catalina.home}/webapps/my-web-app-1/-" {
   permission java.net.SocketPermission "localhost", "resolve";
   permission java.io.FilePermission "…/resources/good-scripts*", "execute";
   permission java.net.SocketPermission "127.0.0.1:32768", "connect,resolve";
   permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
   permission java.io.SerializablePermission "enableSubclassImplementation";
   permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
   };
```

Permission Target Name	What the Permission Allows	Risks of Allowing this Permission
suppressAccessChecks	ability to access fields and invoke methods in a class. Note that this includes not only public, but protected and private fields and methods as well.	This is dangerous in that information (possibly confidential) and methods normally unavailable would be accessible to malicious code.

Changes coming down the pike…

- Java 9
  - Modularization
  - Improved encapsulation
  - Finer control over package access.

Meanwhile

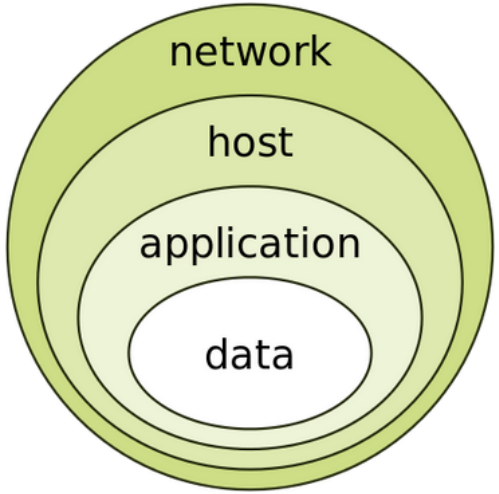
- What should we do?

Defense in depth [edit]

https://en.wikipedia.org/wiki/Information_security

Main article: Defense in depth (computing)

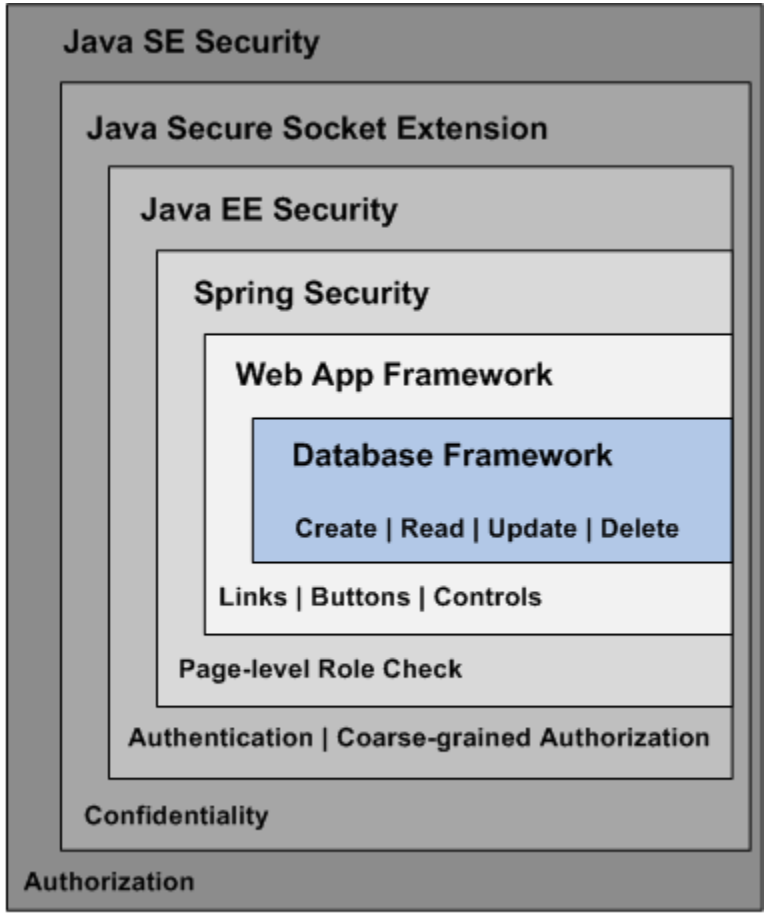
Information security must protect information throughout the life span of the information, from the initial creation of the information on through to the final disposal of the information. The information must be protected while in motion and while at rest. During its lifetime, information may pass through many different information processing systems and through many different parts of information processing systems. There are many different ways the information and information systems can be threatened. To fully protect the information during its lifetime, each component of the information processing system must have its own protection mechanisms. The building up, layering on and overlapping of security measures is called **defense in depth**. In contrast to a metal chain, which is famously only as strong as its weakest link, the defense-in-depth aims at a structure where, should one defensive measure fail, other measures will continue to provide protection.



The onion model of defense in depth

Java Web Security Layers

1. Java SE Security
2. Java Secure Socket Extension (JSSE)
3. Java EE Security
4. Spring Security
5. Web App Framework
6. Database Framework



Each with a specific purpose

1. Java SE Security ---------- principle of least privilege
2. JSSE ---------- private conversations
3. Java EE Security ---------- deadbolt on front door
4. Spring Security ---------- locks on room doors
5. Web App Framework - locks on equipment in rooms
6. Database Functions ---- content filtering

Two Areas of Access Control

1. Java and Spring Role Declarative checks

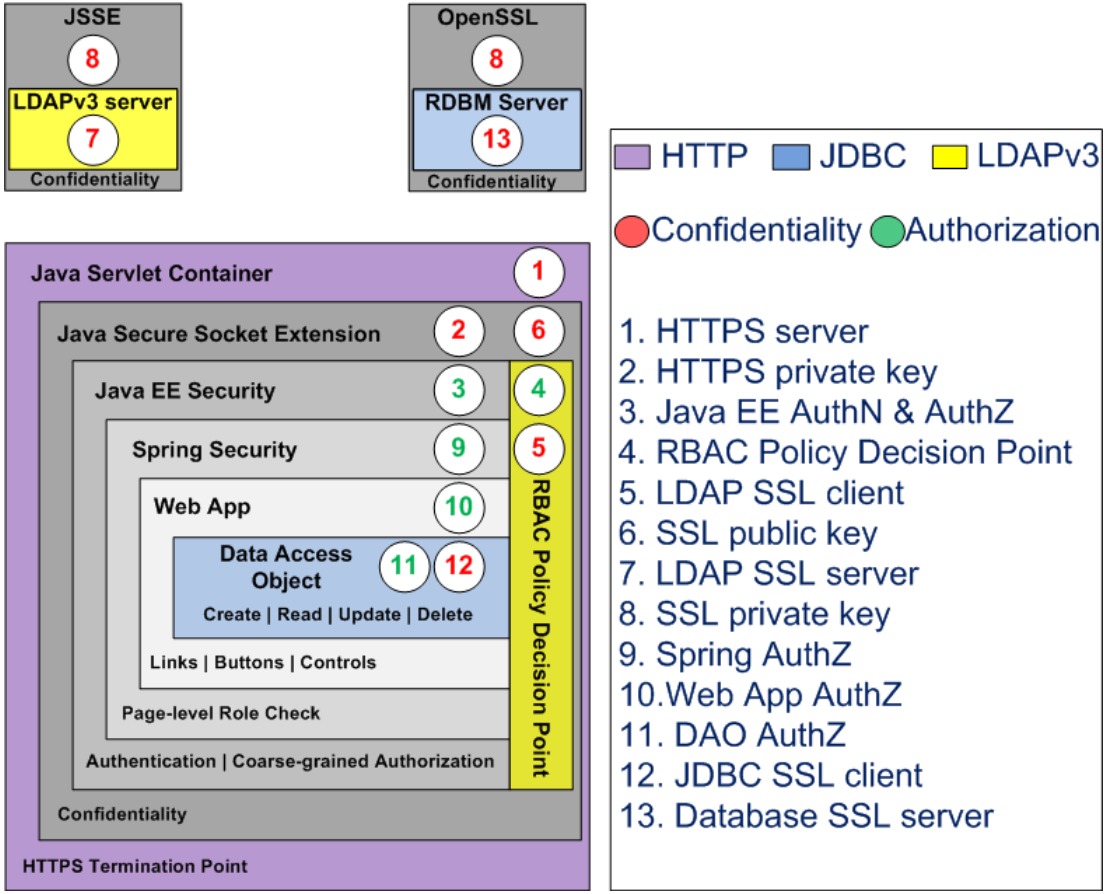
2. RBAC Permission Programmatic checks

Example #2

Apache

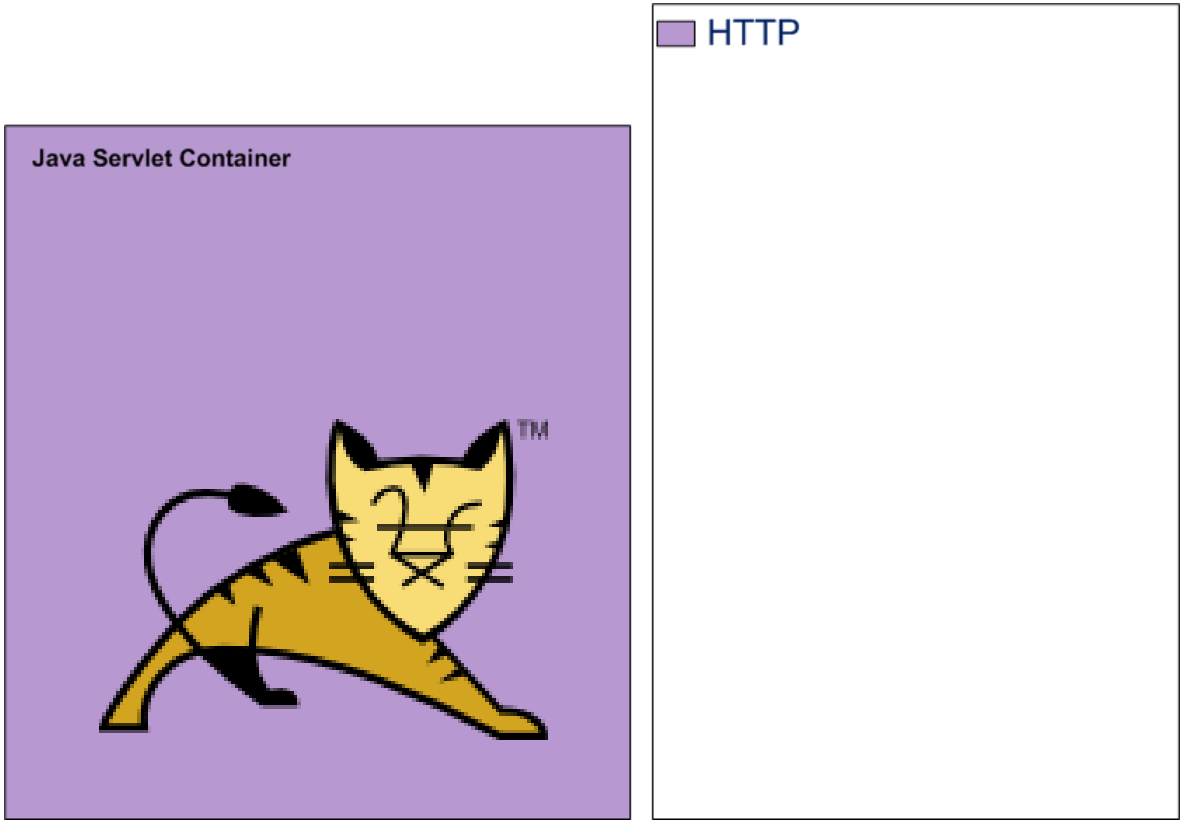
Fortress

Demo



https://github.com/shawnmckinney/apache-fortress-demo

Start with Tomcat Servlet Container

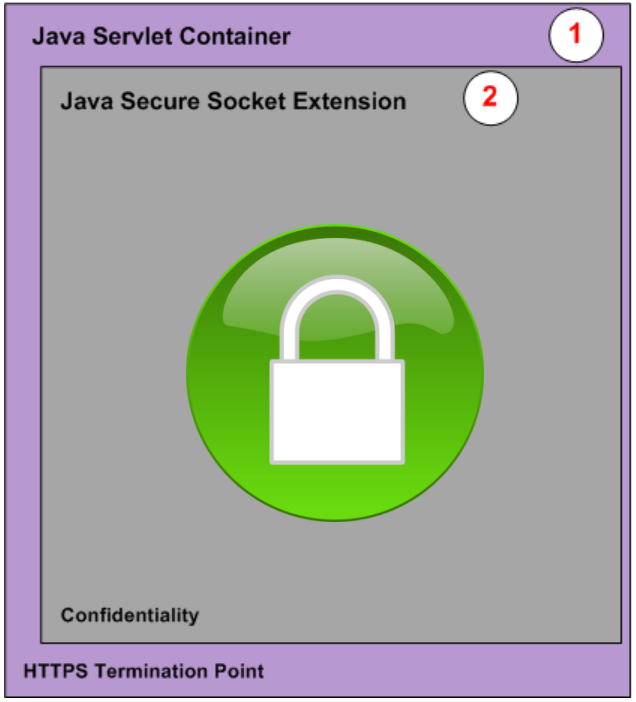


1 & 2. Enable HTTPS

ssssh!!!

1. Update the Server.xml

2. Add private key



HTTP

Confidentiality

1. HTTPS server
2. HTTPS private key

Enable Tomcat TLS

1. Generate keystore with private key (Steps 1 - 5):

http://shawnmckinney.github.io/apache-fortress-demo/apidocs/doc-files/keys.html

2. Add the following to server.xml:

```
<Connector port="8443" maxThreads="200" scheme="https"
    secure="true"
   SSLEnabled= "true“
   keystoreFile=  “/path/mykeystore”
   keystorePass= “*******”
   clientAuth="false" sslProtocol="TLS"/>
```

http://shawnmckinney.github.io/apache-fortress-demo/apidocs/doc-files/apache-tomcat-ssl.html

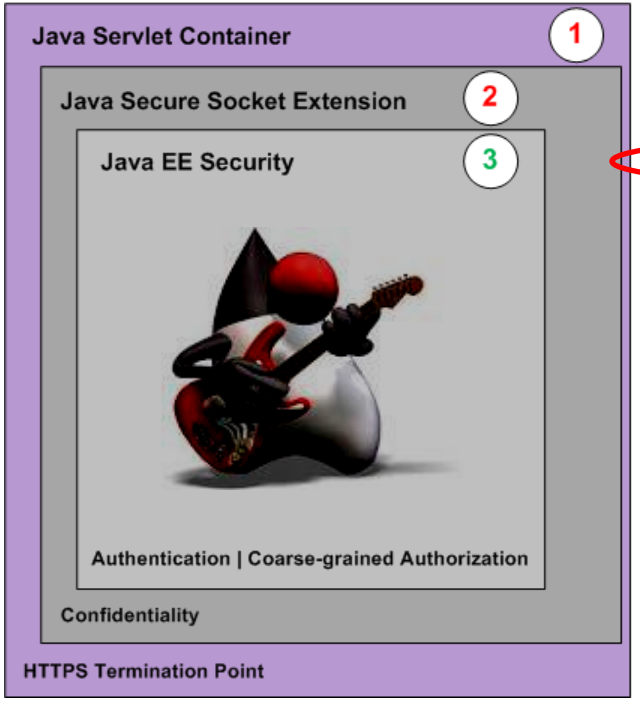
3. Enable Java EE Security the deadbolt

a. Update web.xml

b. Drop the proxy jar

c. Add context.xml

d. Add fortress to pom.xml



Current Specs for Java EE Security

1. JSR-196 – JASPIC - AuthN
2. JSR-115 – JAAC - AuthZ
3. JSR-375 – JavaEE Security API

Enable Java EE Security Realm

Add to App's Web.xml

```
<security-constraint>
  <display-name>My Project Security Constraint</display-name>
  <web-resource-collection>
    <web-resource-name>Protected Area</web-resource-name>
    <url-pattern>/wicket/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>DEMO2_USER</role-name>
  </auth-constraint>
</security-constraint>
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>MySecurityRealm</realm-name>
  <form-login-config>
    <form-login-page>/login/login.html</form-login-page>
```

1. Java EE container protects this URL Automatically.

2. All users must have this role to gain entry.

3. Route un-authN requests to my form.

https://github.com/shawnmckinney/apache-fortress-demo/blob/master/src/main/webapp/WEB-INF/web.xml

Enable Java EE Security Realm

Drop the Fortress Realm Proxy Jar in Tomcat's lib folder:

```
[root@IL1SCOLSP102 lib]# pwd
/usr/local/tomcat7/lib
[root@IL1SCOLSP102 lib]# ls -l fortress*
-rw-r--r-- 1 root root 15119 Aug 29 12:37 fortress-realm-proxy-1.0-RC41-SNAPSHOT.jar
[root@IL1SCOLSP102 lib]#
```

Enable Java EE Security Realm

Add **context.xml** to META-INF folder:

```
<Context reloadable="true">
 <Realm className=
   "org.apache.directory.fortress.realm.tomcat.Tc7AccessMgrProxy"

    defaultRoles="ROLE_DEMO2_SUPER_USER,DEMO2_ALL_PAGES,
                  ROLE_PAGE1, ROLE_PAGE2, ROLE_PAGE3"
/>
</Context>
```

Apache Fortress Tomcat Realm

The set of role candidates eligible to be actived into a session.

https://github.com/shawnmckinney/apache-fortress-demo/blob/master/src/main/resources/META-INF/context.xml

Enable RBAC Policy Decision Point

Add Fortress Dependency to web app's pom.xml:

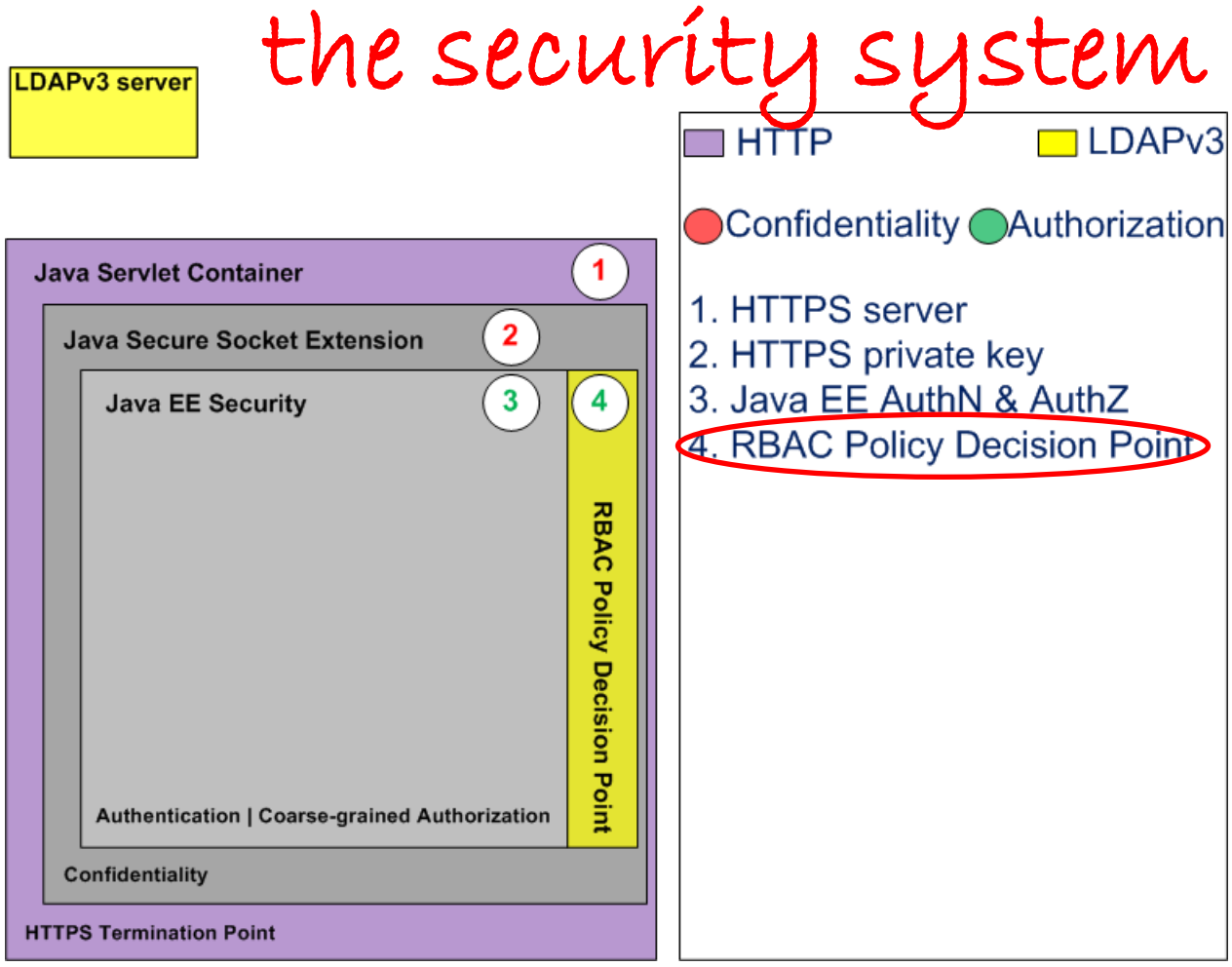
```
<dependency>
  <groupId>org.apache.directory.fortress</groupId>
  <artifactId>
```

fortress-realm-impl

```
[root@IL1SCOLSP102 lib]# pwd
/usr/local/tomcat7/webapps/apache-fortress-demo/WEB-INF/lib
[root@IL1SCOLSP102 lib]# ls -l fortress*
-rw-r--r-- 1 root root 502112 Aug 30 06:55 fortress-core-1.0-RC41-SNAPSHOT.jar
-rw-r--r-- 1 root root  22005 Aug 29 12:20 fortress-realm-impl-1.0-RC41-SNAPSHOT.jar
-rw-r--r-- 1 root root 789927 Aug 29 12:40 fortress-web-1.0-RC41-SNAPSHOT-classes.jar
[root@IL1SCOLSP102 lib]#
```

4. Setup RBAC PDP

Policy Decision Point
a. Install
b. Configure
c. Use



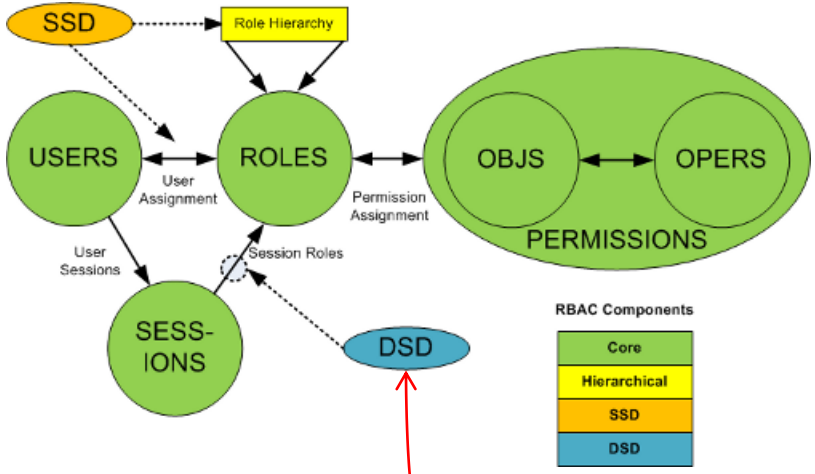
Use ANSI RBAC INCITS 359 Specification

RBAC0:
- Users, Roles, Perms, Sessions

RBAC1:
- Hierarchical Roles

RBAC2:
- Static Separation of Duties

RBAC3:
- Dynamic Separation of Duties



Today we demo this

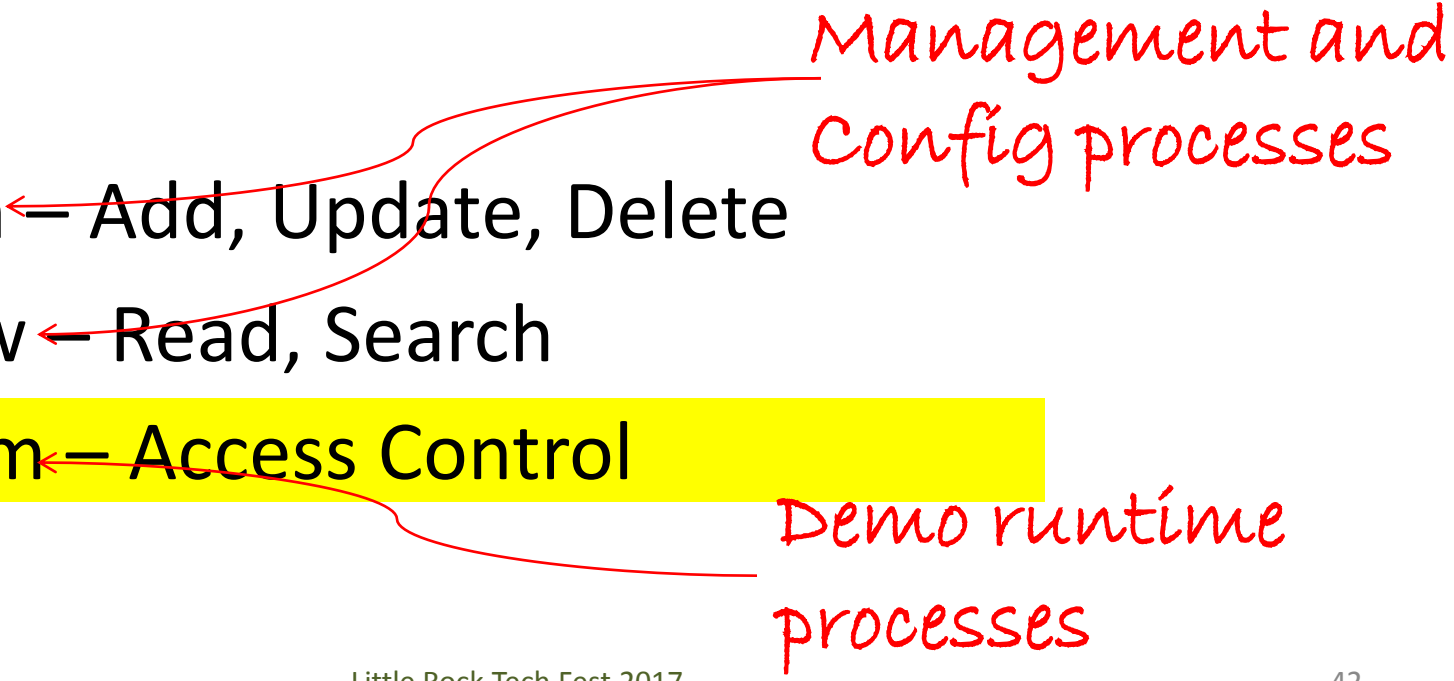
Use RBAC Object Model

Six basic elements:

1. **User** – human or machine entity
2. **Role** – a job function within an organization
3. **Object** – maps to system resources
4. **Operation** – executable image of program
5. **Permission** – approval to perform an Operation on one or more Objects
6. **Session** – contains set of activated roles for User

Use RBAC Functional Model

APIs form three standard interfaces:



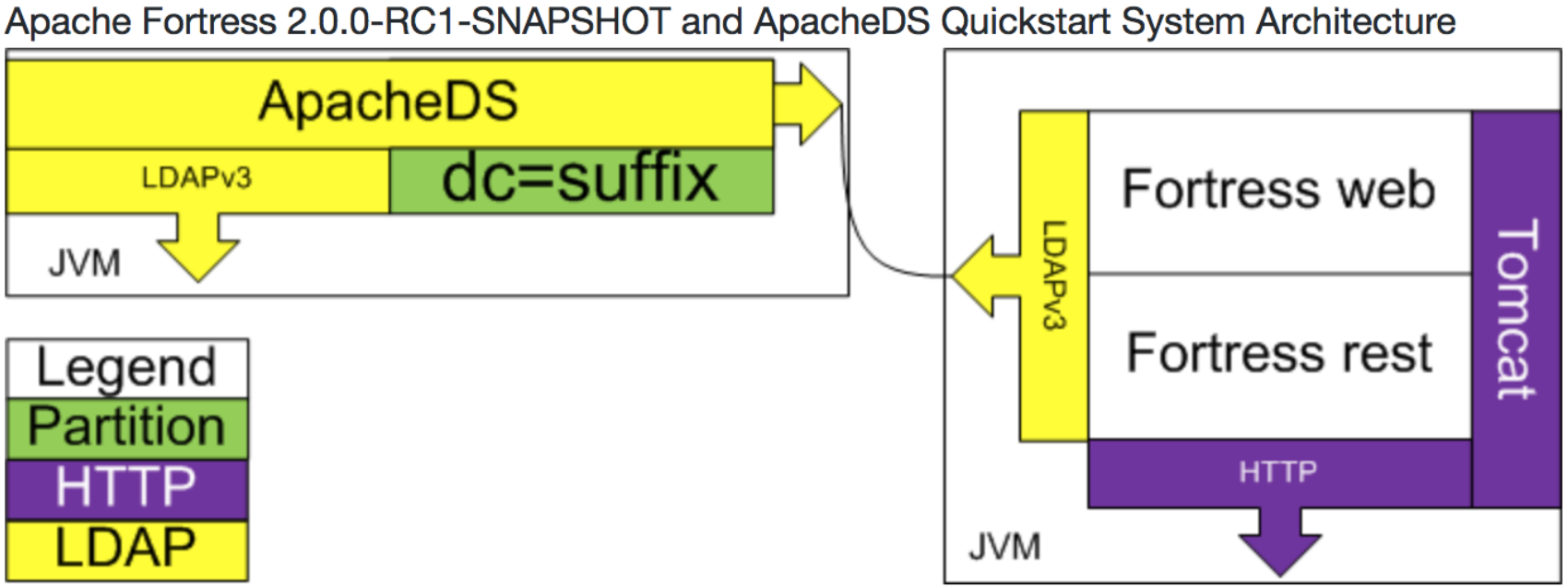
Use RBAC Functional Model

System Manager APIs:

http://directory.apache.org/fortress/gen-docs/latest/apidocs/org/apache/directory/fortress/core/impl/AccessMgrImpl.html

1. createSession – authenticate, activate roles
2. checkAccess – permission check
3. sessionPermissions – all perms active for user
4. sessionRoles – return all roles active
5. addActiveRole – add new role to session
6. dropActiveRole – remove role from session

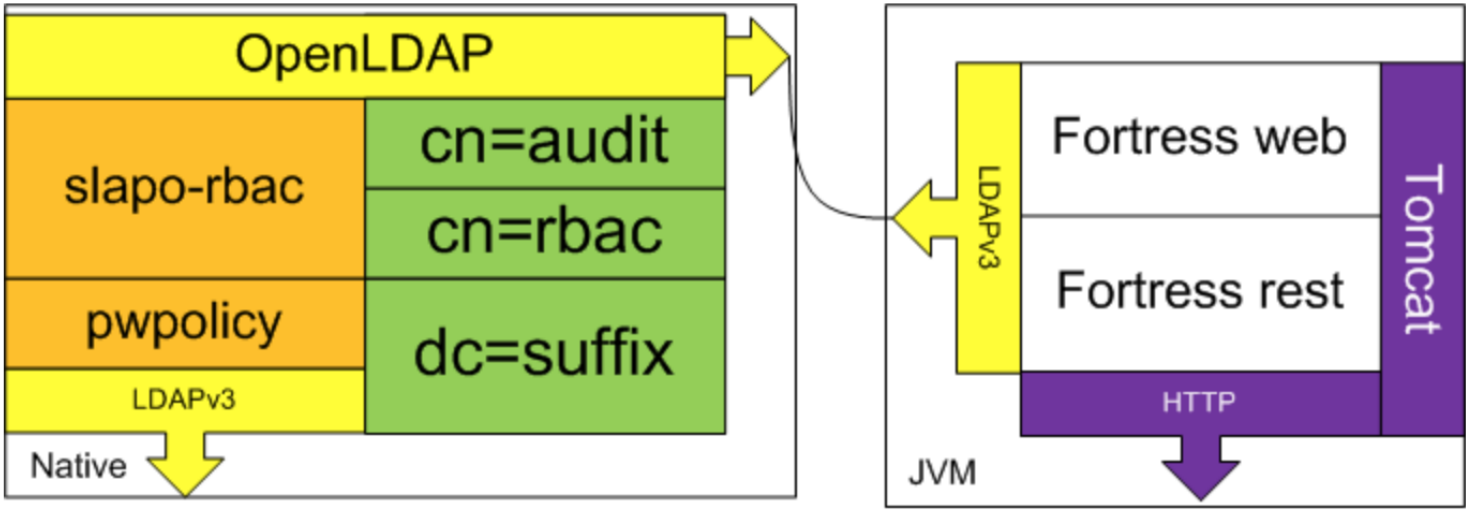
ApacheDS & Fortress QUICKSTART



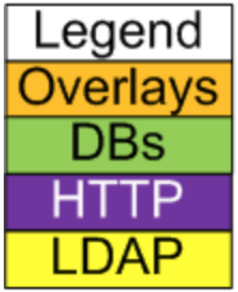
https://github.com/apache/directory-fortress-core/blob/master/README-QUICKSTART-APACHEDS.md

Or OpenLDAP & Fortress QUICKSTART

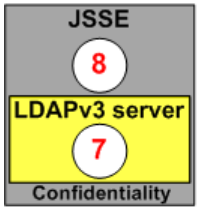
Apache Fortress 2.0.0-RC2 and OpenLDAP Quickstart System Architecture



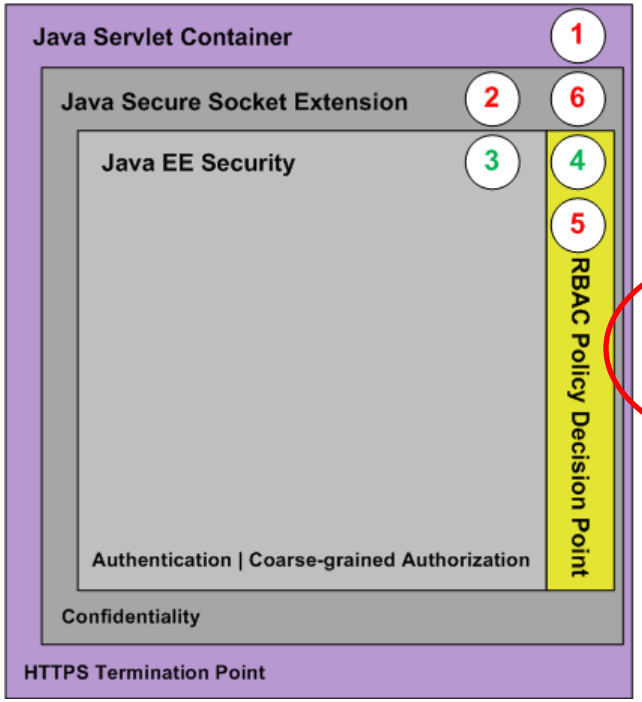
https://github.com/apache/directory-fortress-core/blob/master/README-QUICKSTART-SLAPD.md

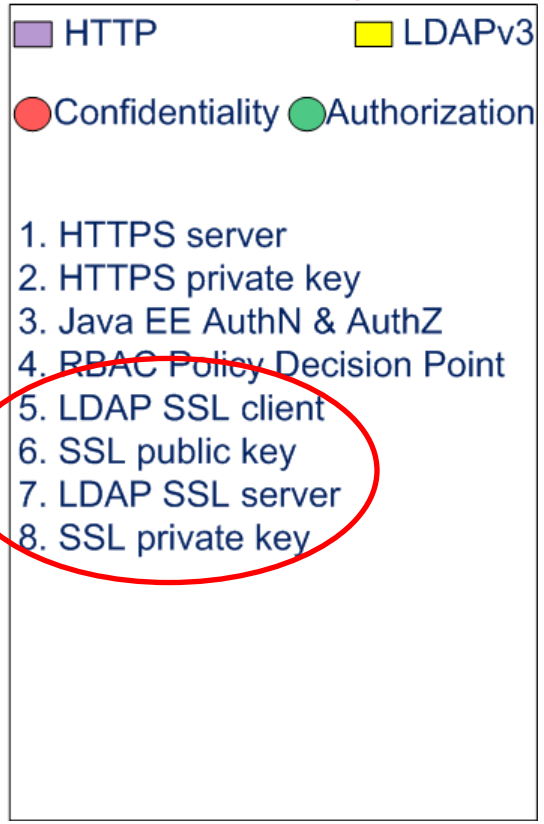


5 – 8 Enable LDAP SSL



confidentiality





Enable LDAP SSL Client

1. Import public key to java truststore: http://shawnmckinney.github.io/apache-fortress-demo/apidocs/doc-files/keys.html

2. Add to fortress.properties

```
host=ldap-server-domain-name.com
port=636
enable.ldap.ssl=true
trust.store=mytruststore
trust.store.password=changeit
trust.store.onclasspath=true
```

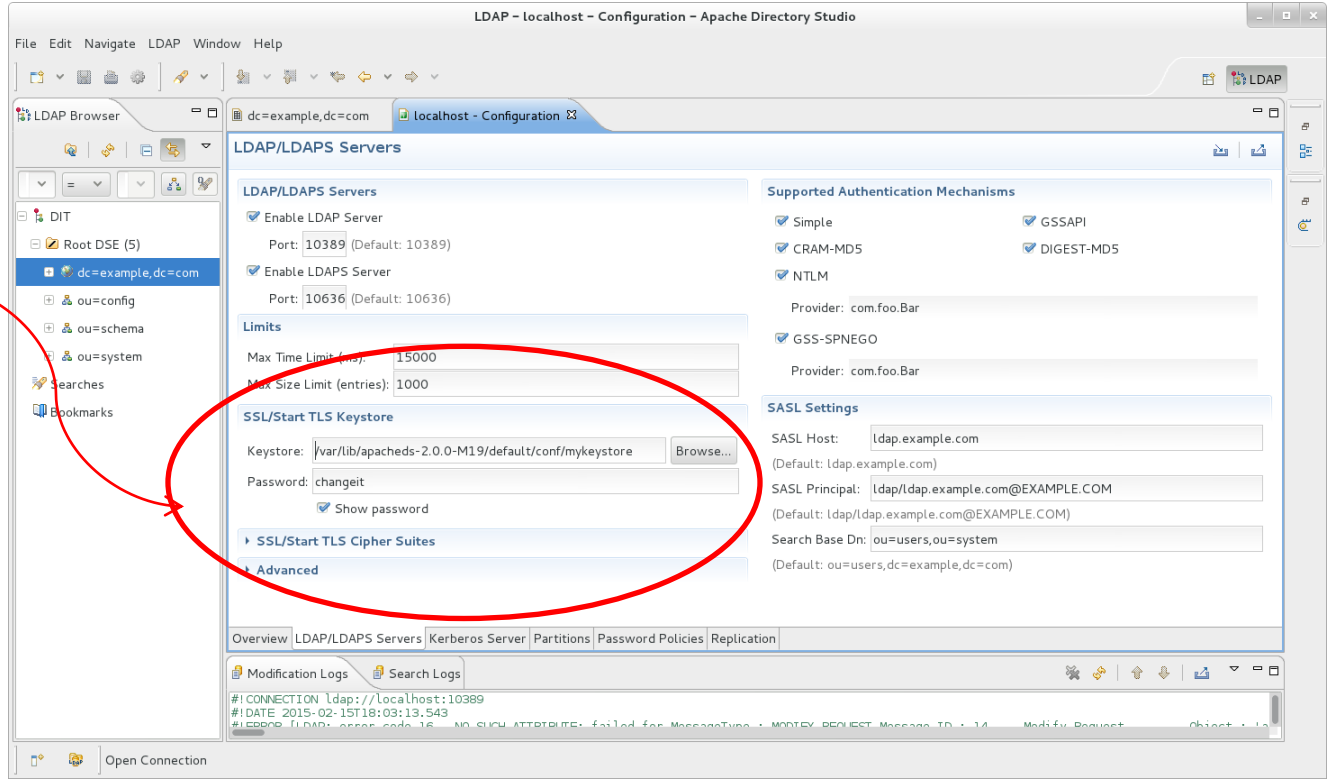
common name in server cert

can be found on classpath

Enable ApacheDS LDAP SSL Server

1.Import keystore with Apache Directory Studio

2.Restart ApacheDS Server



http://shawnmckinney.github.io/apache-fortress-demo/apidocs/doc-files/apache-directory-ssl.html

Or Enable OpenLDAP SSL Server

Add locations of crypto artifacts to slapd server config:

TLSCACertificateFile /path/to/my/ca-certificate

TLSCertificateFile /path/to/my/server-certificate

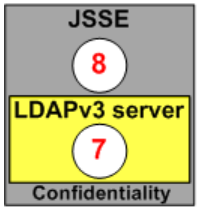
TLSCertificateKeyFile /path/to/my/private-key

http://shawnmckinney.github.io/apache-fortress-demo/apidocs/doc-files/openldap-ssl.html

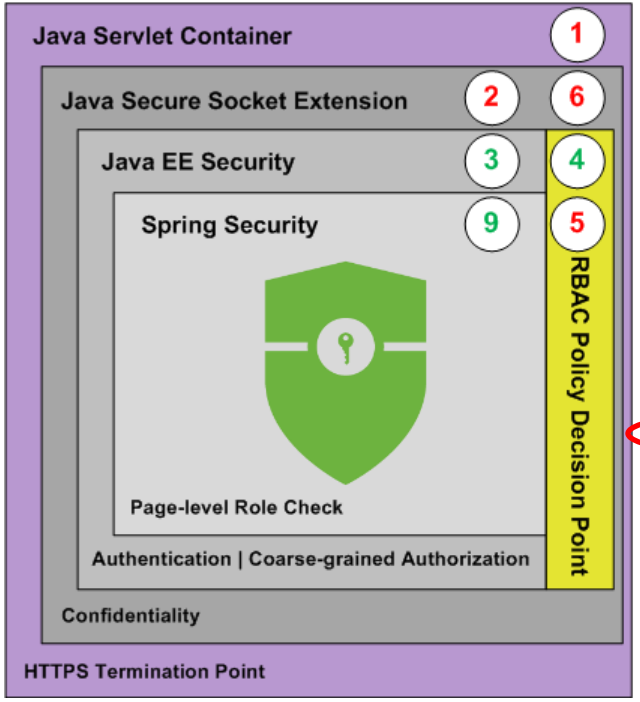
9. Enable Spring Security

a. Authorization

b. Role mapping



locks on the rooms



Enable Spring Security

Add dependencies to pom:

```
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId> spring-security-core </artifactId>
   <version>4.1.3.RELEASE</version>
</dependency>
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId> spring-security-config </artifactId>
   <version>4.1.3.RELEASE</version>
</dependency>
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId> spring-security-web </artifactId>
   <version>4.1.3.RELEASE</version>
</dependency>
```

Add the Spring Context File to App

Enable Spring's context file via web app's web.xml file:

```
<context-param>
 <param-name>contextConfigLocation</param-name>
 <param-value>
    classpath:applicationContext.xml
 </param-value>
</context-param>
```

Enable Spring Security Interceptor

```
<bean id="fsi“=
"org.springframework.security.web.access.intercept.FilterSecurityInterceptor">
  <property name="authenticationManager" ref="authenticationManager"/>
  <property name="accessDecisionManager" ref="httpRequestAccessDecisionManager"/>
  <property name="securityMetadataSource">
   <sec:filter-security-metadata-source use-expressions="false">
    <sec:intercept-url pattern=
      "…/com.mycompany.page1"
      access="ROLE_PAGE1"
    />
   </sec:filter-security-metadata-source>
  </property>
</bean>
```

page-level authorization (declarative)

By default name must contain ROLE_

Role Mapping

Role Propagation between Java EE & Spring Security

Spring Security uses PreAuthenticatedAuthentication filter to get java EE role mappings.

From the applicationContext.xml:

```
<bean id="preAuthenticatedAuthenticationProvider”
 class="org.springframework.security.web.authentication.preauth.
 PreAuthenticatedAuthenticationProvider">
  <property name="preAuthenticatedUserDetailsService"  ref="preAuthenticatedUserDetailsService"/>
</bean>
…
```

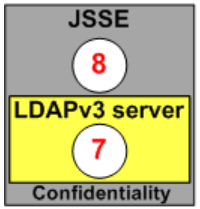
Role Mapping

Share Roles Between Java EE and Spring

Complete list of eligible roles found in app's web.xml:

```
<!-- Declared in order to be used by Spring Security -->
<security-role>
  <role-name>ROLE_DEMO2_SUPER_USER</role-name>
</security-role>
<security-role>
  <role-name>ROLE_PAGE1</role-name>
</security-role>
<security-role>
  <role-name>ROLE_PAGE2</role-name>
</security-role>
<security-role>
  <role-name>ROLE_PAGE3</role-name>
</security-role>
```

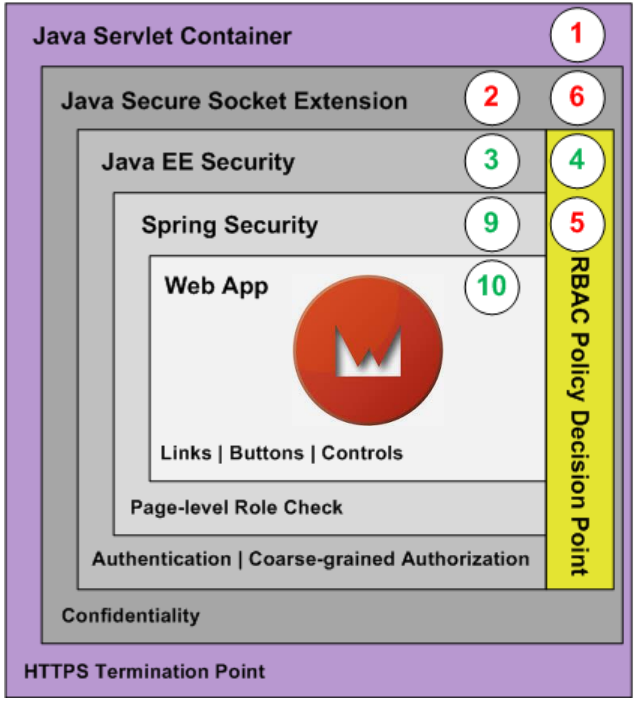
10. Web App Authorization

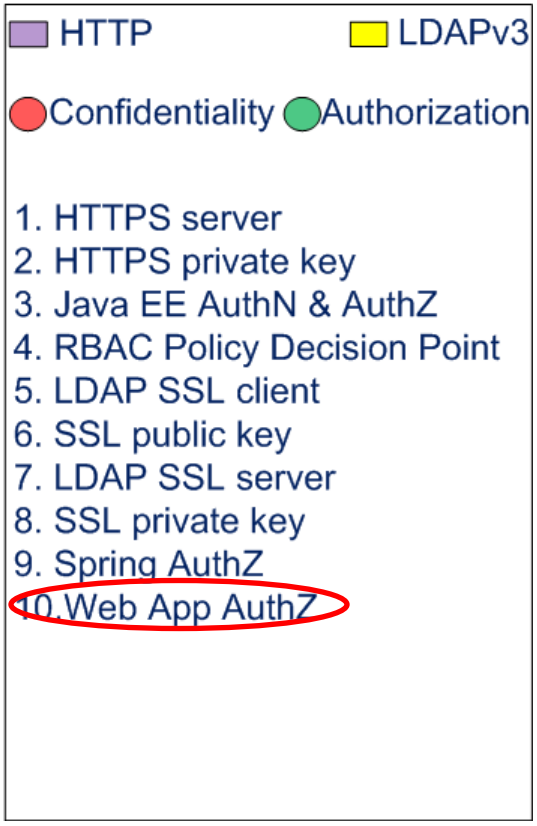


locks on equipment

Add fine-grained checks:

a. Page links
b. Buttons
c. Other controls





Inject Fortress APIs via Spring Beans

Enable Fortress RBAC Spring Beans in applicationContext.xml:

```
<bean id= “accessMgr”
  class= "org.apache.directory.fortress.core.AccessMgrFactory"
  scope="prototype"
  factory-method="createInstance">
  <constructor-arg value="HOME"/>
</bean>
```

Share the Session with Tomcat

Session Propagation between Tomcat, Fortress and Web app:

1. The Fortress Tomcat Realm creates the session after user successfully authenticates. It serializes the data and stores inside a principal object.

2. Tomcat returns the serialized principal to Web app on request:

```
String szPrin = servletRequest.getUserPrincipal().toString();
```

<- Standard Java api

3. Next deserialize the java security principal into a 'Fortress' session:

```
Session ftSess = j2eePolicyMgr.deserialize( szPrin );
```

<- Fortress Realm api

4. Store the Fortress session into an HTTP session object for later usage:

```
myAppFw.setSession( ftSess );
```

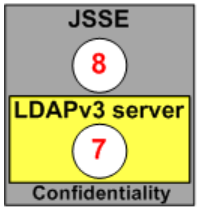
<- Web app's own api

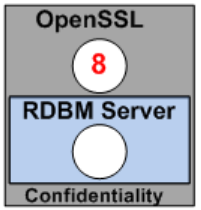
Add Web Framework Security

```
public class Page1 extends MyBasePage
{
  Add( new SecureIndicatingAjaxButton( "Page1", "Add" )
  {
    @Override
    protected void onSubmit( ... )
    {
      if( checkAccess( customerNumber ) )
      {
         // do something here:
      }
      else
      {
         target.appendJavaScript( ";alert('Unauthorized');" );
      }
    }
  });
```

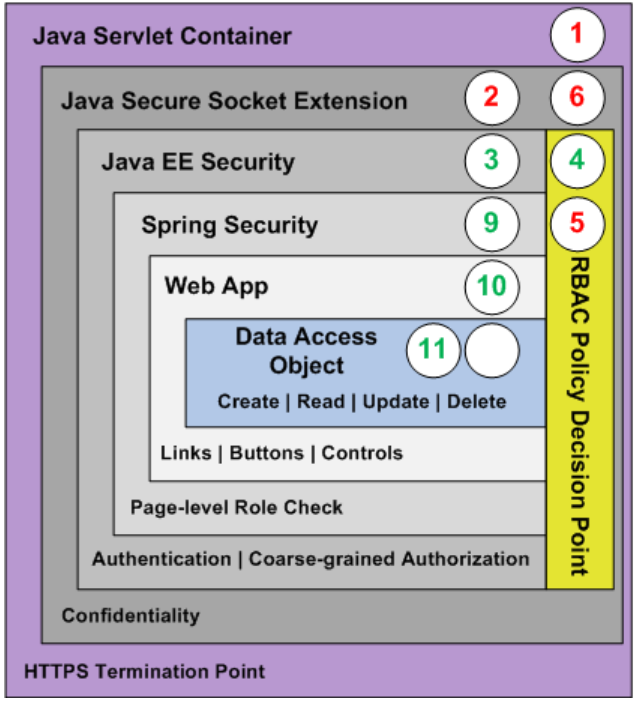
fine-grained authorization (programmatic)

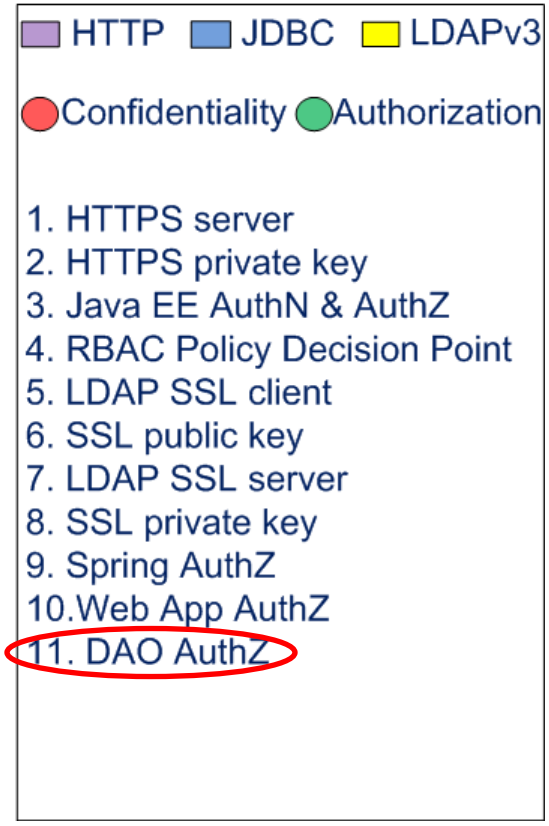
11. DAO Authorization





filtering





Add fine-grained Checks to:

a. Create
b. Read
c. Update
d. Delete

symas

Add Security Aware DAO components

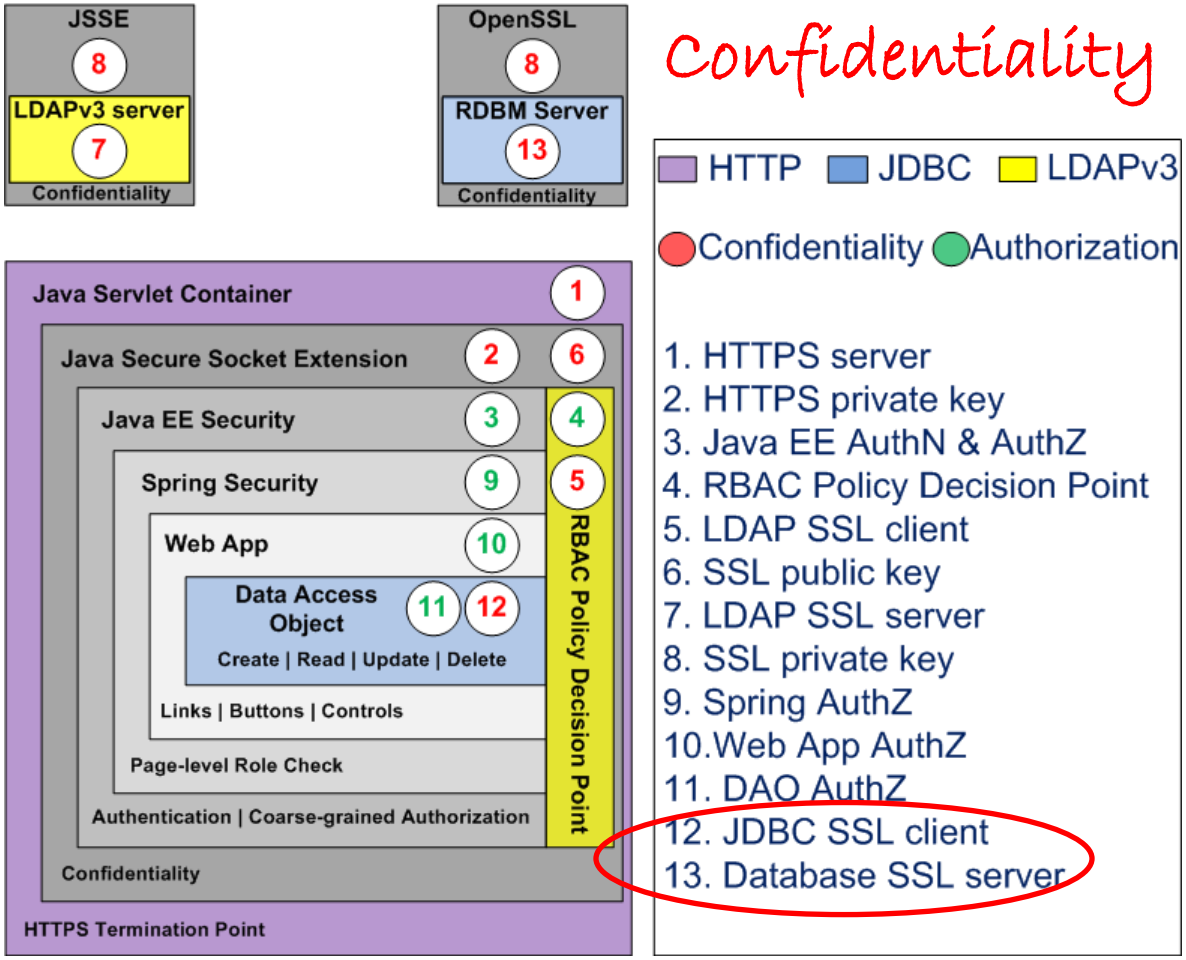
```
public class Page1DaoMgr implements Serializable
{…

  public Page1EO updatePage1( Page1EO entity )
  if(checkAccess("Page1","Update",entity.getCust()))
  {
    // Do normal DAO.update stuff here...
  }
  else
   throw new RuntimeException("Unauthorized");
  ...
  return entity;
  }
…
}
```

fine-grained authorization (programmatic)

12, 13. Enable DB SSL

12. Client
 a. public key
 b. config

13. Server
 a. private key
 b. config



Enable JDBC SSL Client

Add to fortress.properties of Web app:

```
trust.store=/path/mytruststore
trust.store.onclasspath=false
```

must be found on file path

```
# These are the JDBC configuration params for
  MyBatis DAO connect to MySQL database
  example:
database.driver=com.mysql.jdbc.Driver
database.url= db-domain-name.com:3306/
  jdbc:mysql://demoDB ?useSSL=true&amp;requireSSL=true
```

Enable MySQL SSL Server

Add to MySQL my.cnf the server's keys:

```
ssl-ca=/path/ca-cert.pem
ssl-cert=/path/server-cert.pem
ssl-key=/path/server-key.pem
```

2. Instruct listener to use host name in certificate on server restart:

```
bind-address = db-domain-name.com
```

http://shawnmckinney.github.io/apache-fortress-demo/apidocs/doc-files/mysql.html

Apache Fortress Demo

- Three Pages and Three Customers
- One role for every page to customer combo
- Users may be assigned to one or more roles
- One and only one role may be activated

Pages	Customer 123	Customer 456	Customer 789
Page One	PAGE1_123	PAGE1_456	PAGE1_789
Page Two	PAGE2_123	PAGE2_456	PAGE2_789
Page Three	PAGE3_123	PAGE3_456	PAGE3_789

User123	Customer 123	Customer 456	Customer 789
Page1	True	False	False
Page2	True	False	False
Page3	True	False	False

User1	Customer 123	Customer 456	Customer 789
Page1	True	True	True
Page2	False	False	False
Page3	False	False	False

User1_123	Customer 123	Customer 456	Customer 789
Page1	True	False	False
Page2	False	False	False
Page3	False	False	False

Testing

- Verify security functionality via automation.
- Otherwise vulnerabilities may still exist in your system.

https://github.com/shawnmckinney/apache-fortress-demo/.../ApacheFortressDemoSeleniumITCase.java

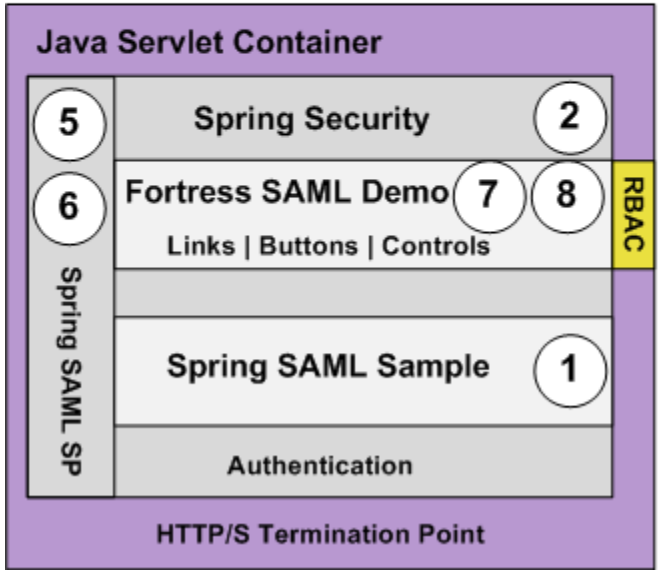
Apache Fortress Demo

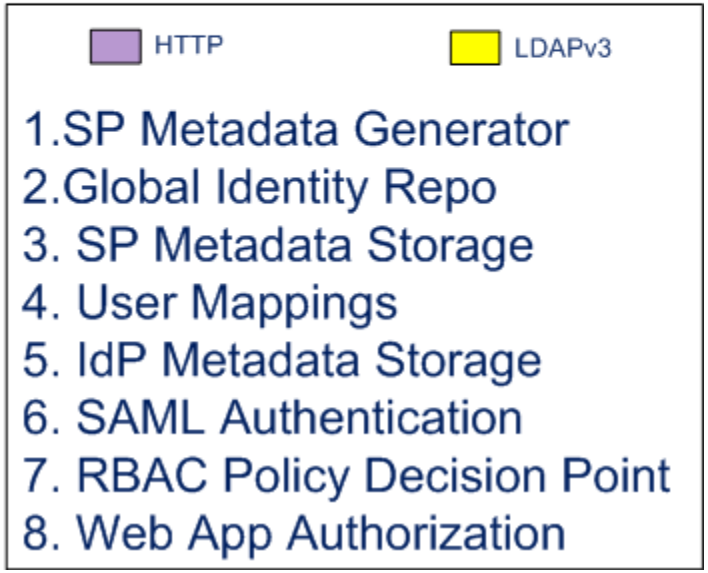
- https://github.com/shawnmckinney/apache-fortress-demo

User Foo	Customer 123	Customer 456	Customer 789
Page1	False	True	True
Page2	True	False	False
Page3	True	False	False

Example #3

Fortress SAML Demo





https://github.com/shawnmckinney/fortress-saml-demo

The Security Layers with SAML

1. Java SE Security
2. JSSE
3. ~~Java EE Security~~ ← Turned off (for now)
4. Spring Security ← Deadbolt is now here
5. Web App Framework
6. Database Functions

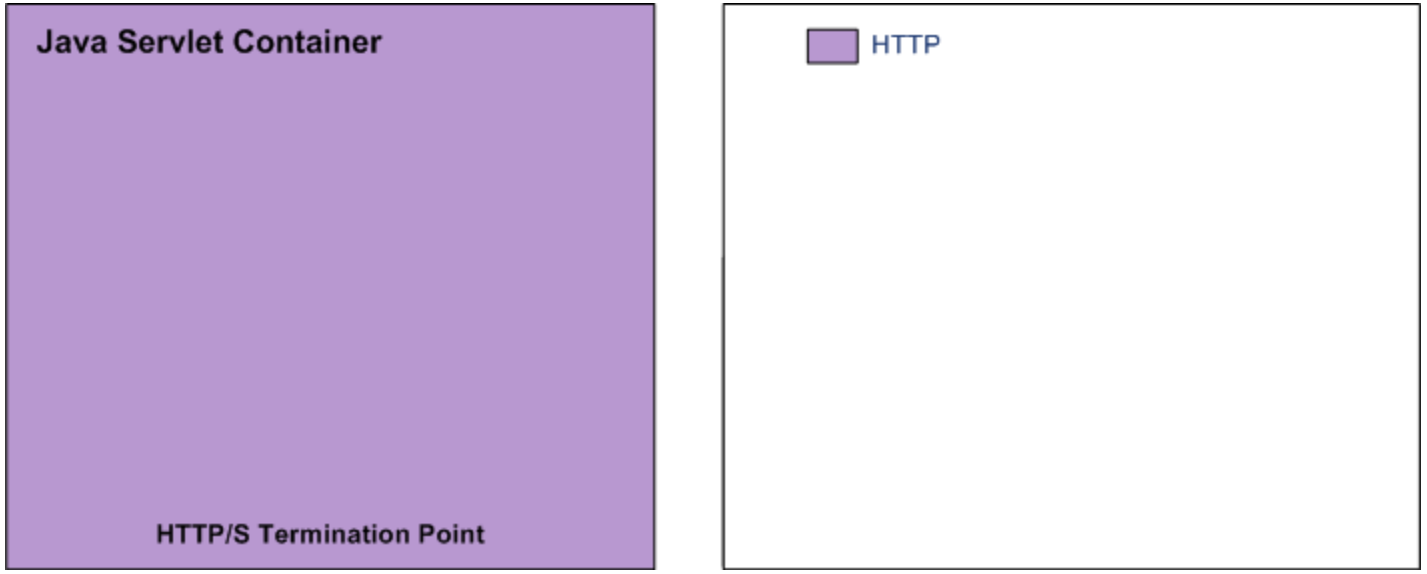
Not much to change

Two Areas of Access Control

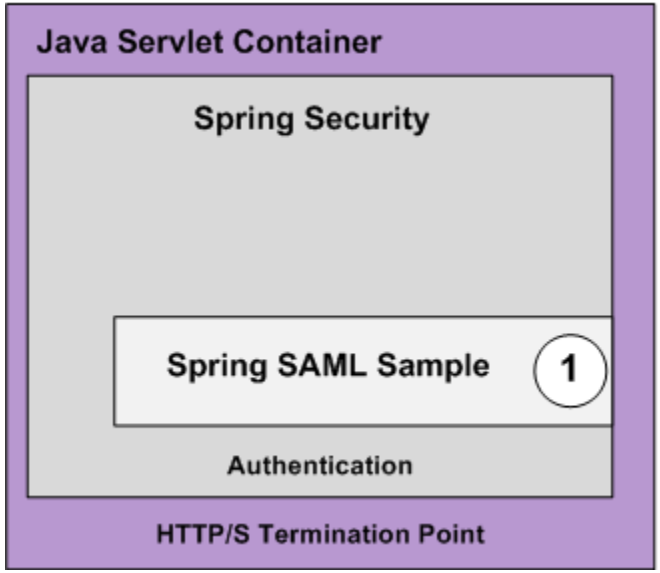
1. Spring SAML Declarative checks

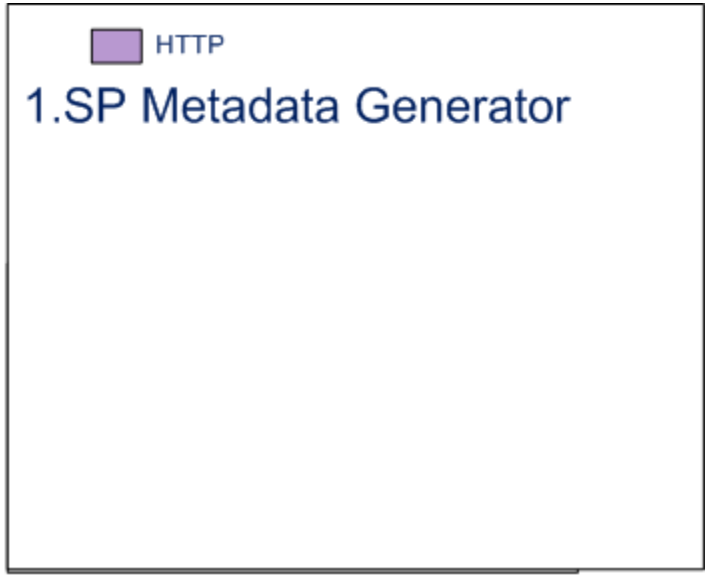
2. RBAC Permission Programmatic checks

Start with Tomcat Servlet Container



1. Deploy the Spring SAML Demo





Get the Spring SAML Demo

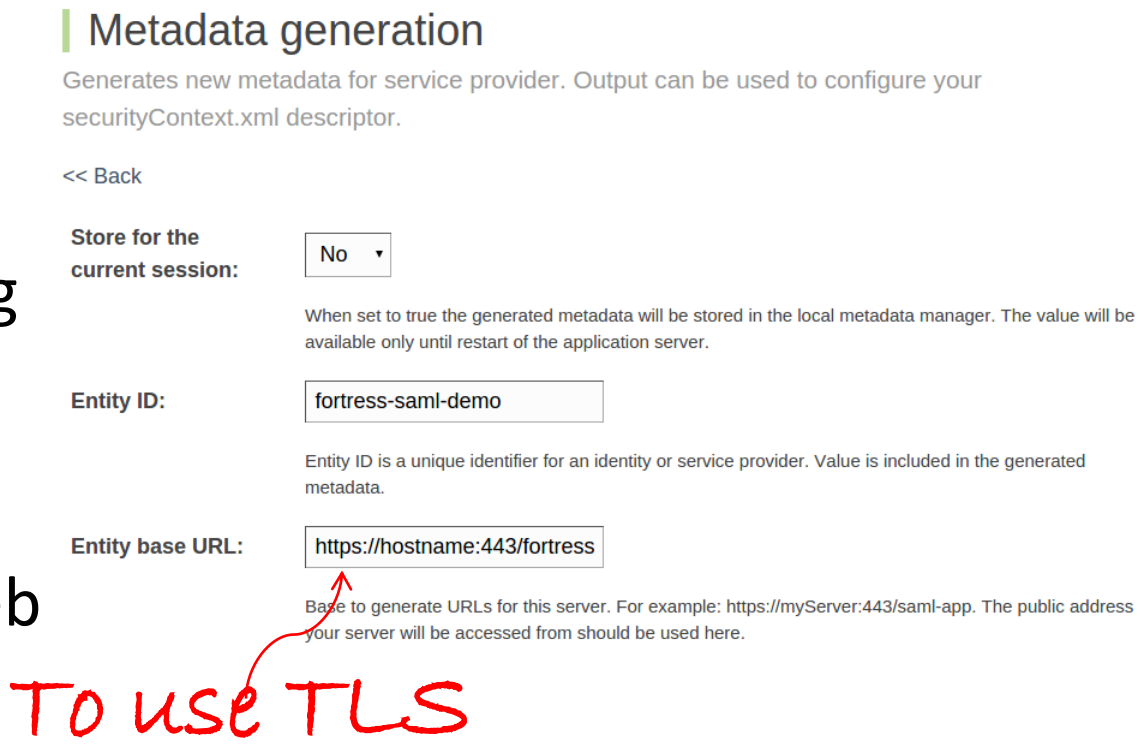
Pick one:

- spring-security-saml - Spring's SAML sample is the first place java developers should look for basic SAML 2.0 programming concepts.
- shibboleth-sample-java-sp - Unicon's sample is where ones goes to understand how to combine Spring SAML's SP with Shibboleth's IdP.

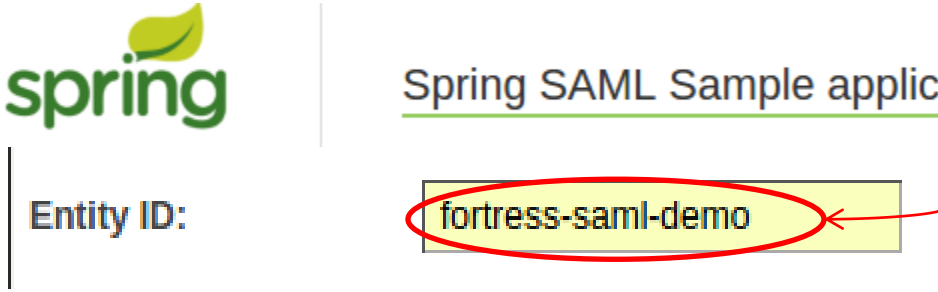
Generate SAML Service Provider Metadata

Matching Fields:

- Entity ID must match Spring config in web app
- Entity base URL must match the web app's URL.



Spring SAML Metadata Generation Tip

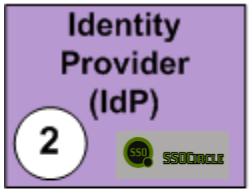


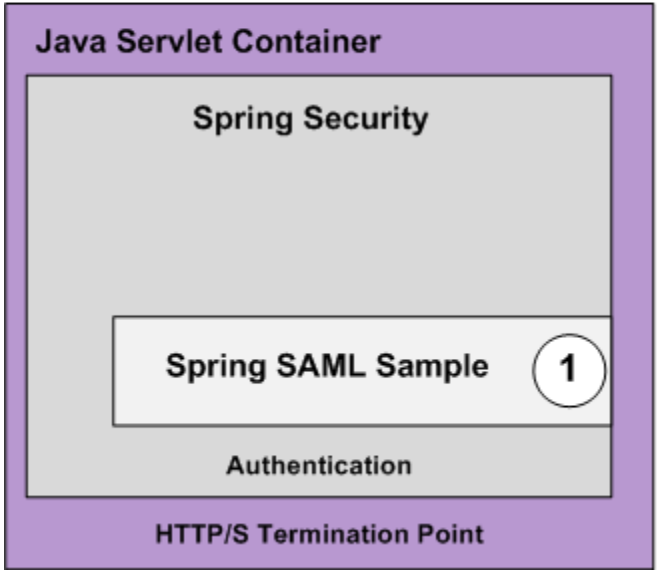
These entityId's must match

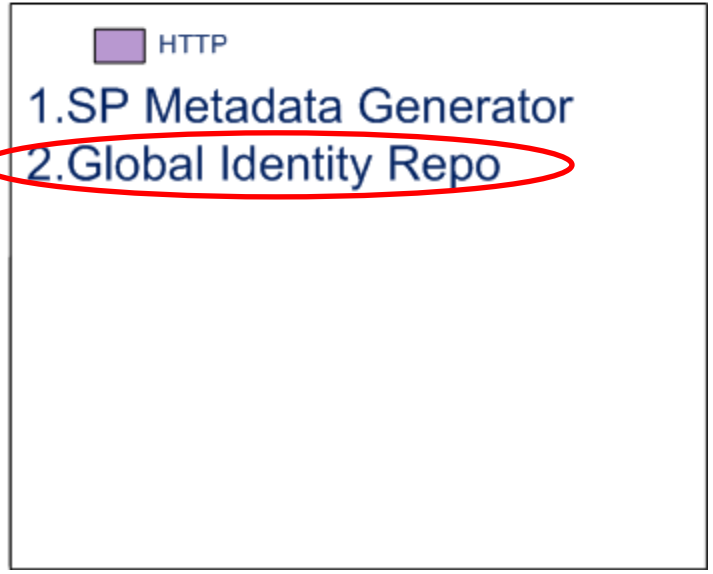
```
<bean id="metadataGeneratorFilter" class="org.springframework…MetadataGeneratorFilter">
 <constructor-arg>
 <bean class="org.springframework…MetadataGenerator”>
  <property name="entityId" value="fortress-saml-demo"/>
 </bean>
 </constructor-arg>
</bean>
```

Bind the service provider with the IdP.

2. Setup Global Identity Provider







Setup SSOCircle SAMLv2.0 IdP

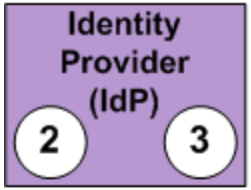
Creating your Identity with SSOCircle (from their website)

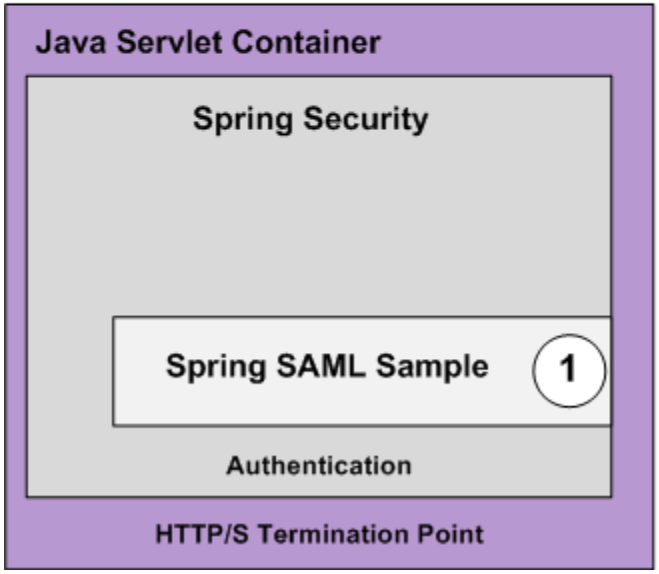
For creating your account you need to follow a few steps:

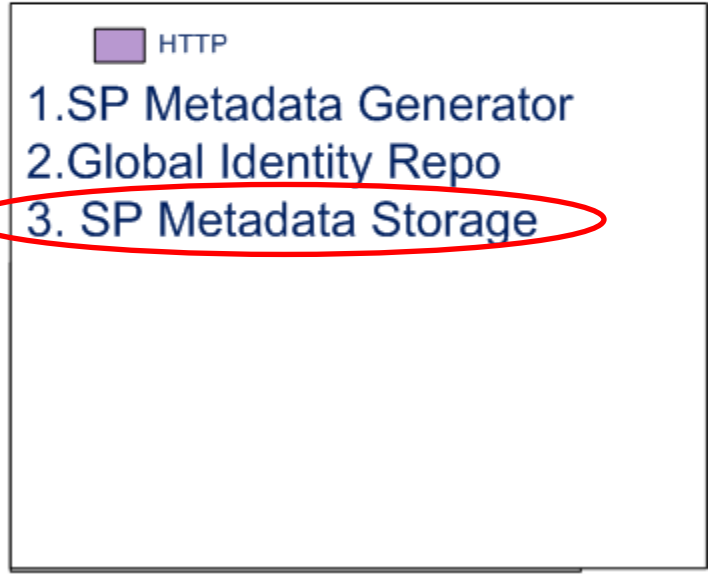
- Register at the SSOCircle SAMLv2.0 Identity Provider
- Provide the required data
- Agree to the Terms of Use
- After successful creation you will receive an email asking for confirmation of your registration. Confirm by navigating to the link supplied in the email.
- Now your account is activated and ready for use.

http://www.ssocircle.com/en/portfolio/publicidp/

3. Import Service Provider Metadata into IdP

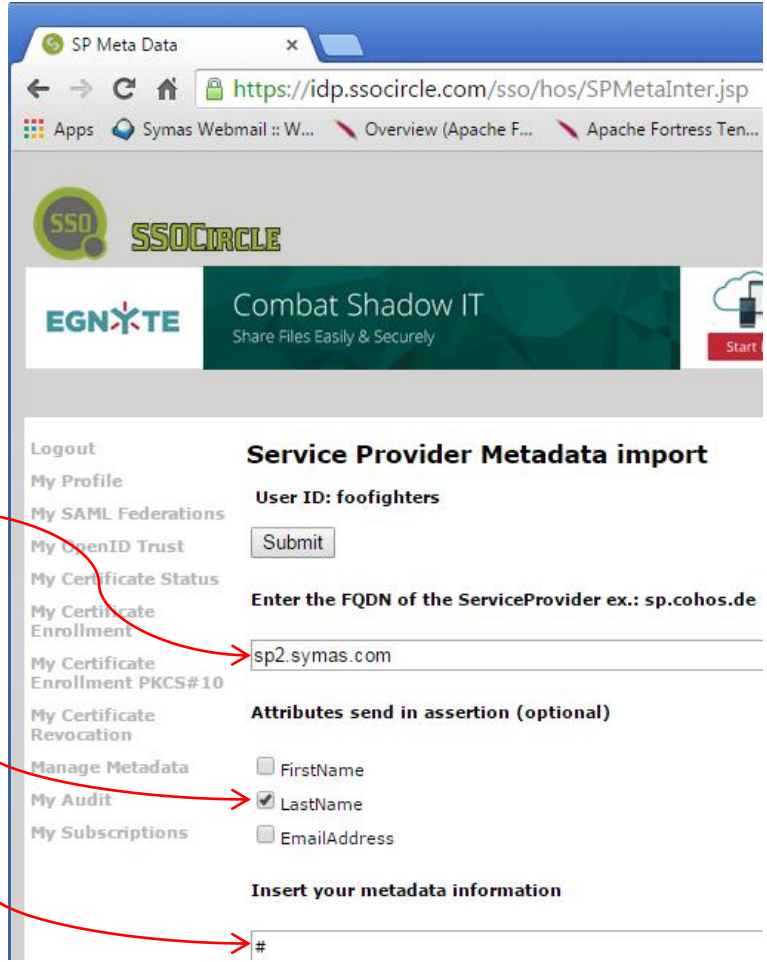






Import SP Metadata

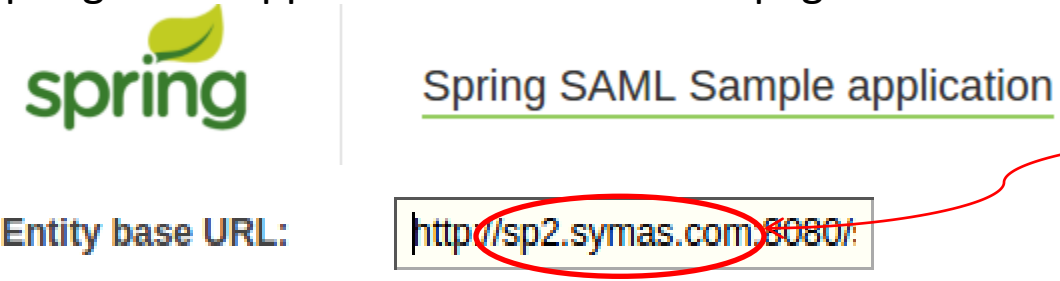
- Logon SSOCircle
- Click on ***Manage Metadata***
- ***FQDN*** must match SP's host name
- Check the ***LastName*** box
- Paste your metadata here



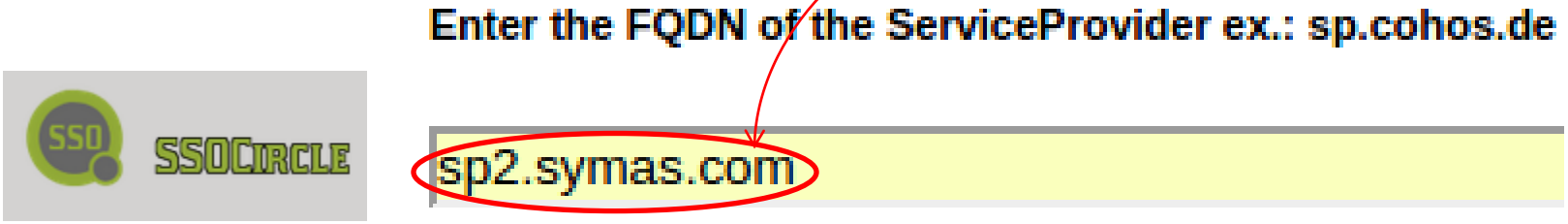
Import SP Metadata Tip

The FQDN matches base url from SP metadata gen step

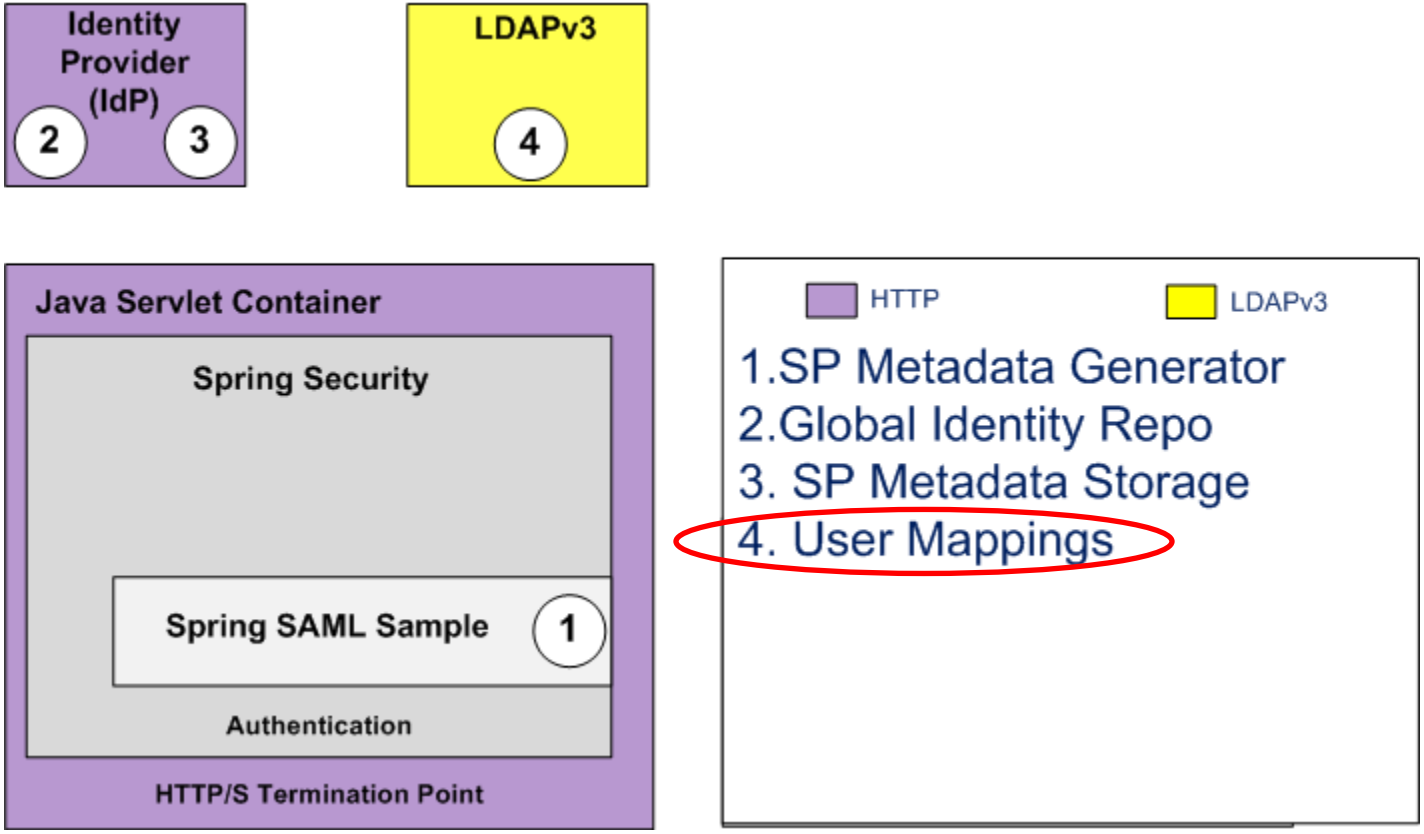
Spring SAML app Metadata Generation page:



SSOCircle Service Provider Metadata Import page:



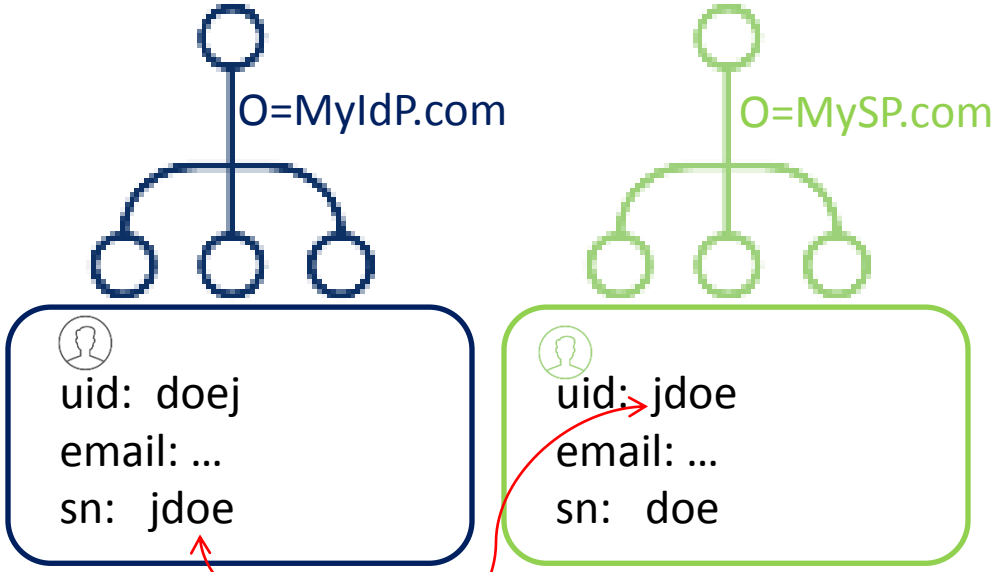
4. IdP and SP User Account Mapping



IdP and SP User Account Mapping

1. Mapping rules are specific to partners.

2. The mapping must be a one-to-one unique pairing.



fortress saml demo maps the sn on the IdP-side with uid field on the SP-side

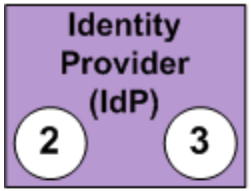
SAML Attribute Statement

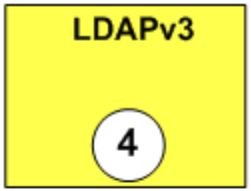
```
<?xml version="1.0" encoding="UTF-8"?><samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol”
Destination="http://sp2.symas.com:8080/fortress-saml-demo/saml/SSO"
…
<saml:AttributeStatement>
…
  <saml:Attribute Name="LastName">
  <saml:AttributeValue …
      xsi:type="xs:string">sam3</saml:AttributeValue>
  </saml:Attribute>
 </saml:AttributeStatement>
…
</samlp:Response>
```

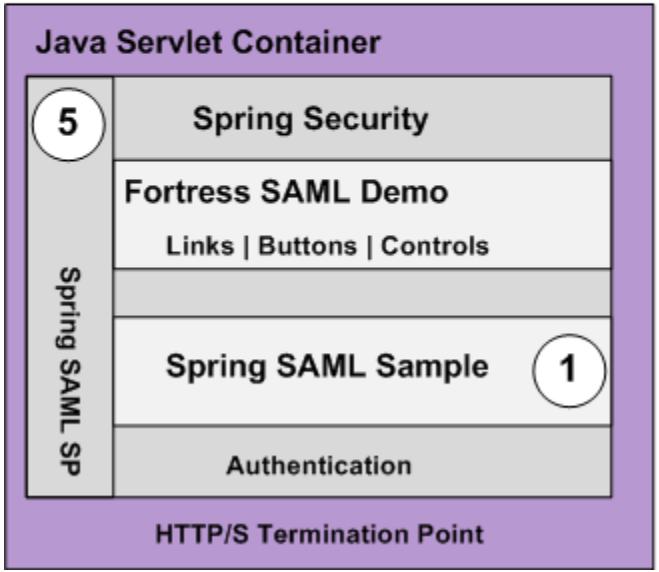
host name entered during SP Metadata import

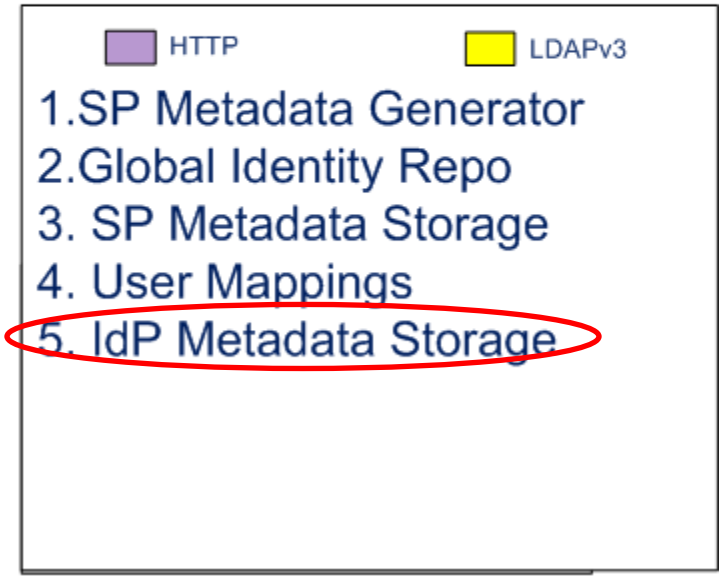
Last Name linked to userid in rbac

5. Load IdP Metadata into Service Provider









Point SP to SAML IdP

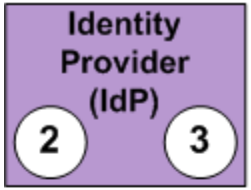
Point to the Identity Provider in securityContext.xml

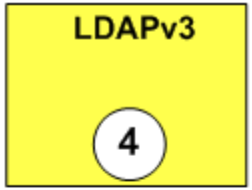
```
<bean id="metadata" class="org.springframework.security.saml.metadata.CachingMetadataManager">
 <constructor-arg>
  <list>
   <bean class="org.opensaml.saml2.metadata.provider.HTTPMetadataProvider">
    <constructor-arg>
     <value type="java.lang.String"
```

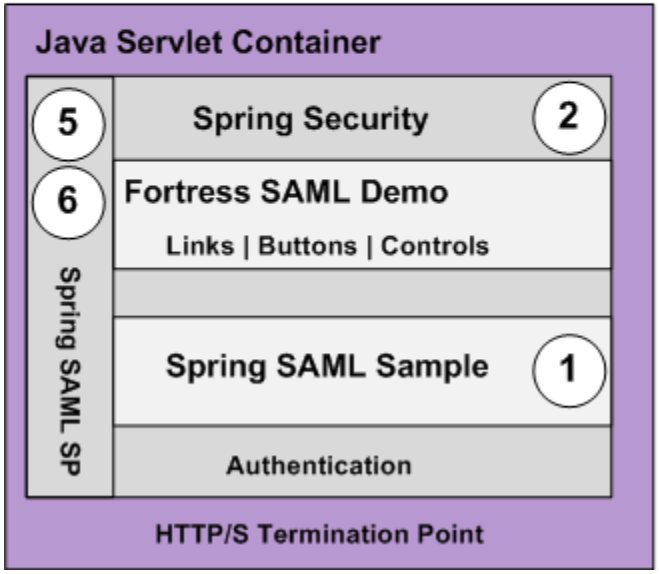
http://idp.ssocircle.com/idp-meta.xml

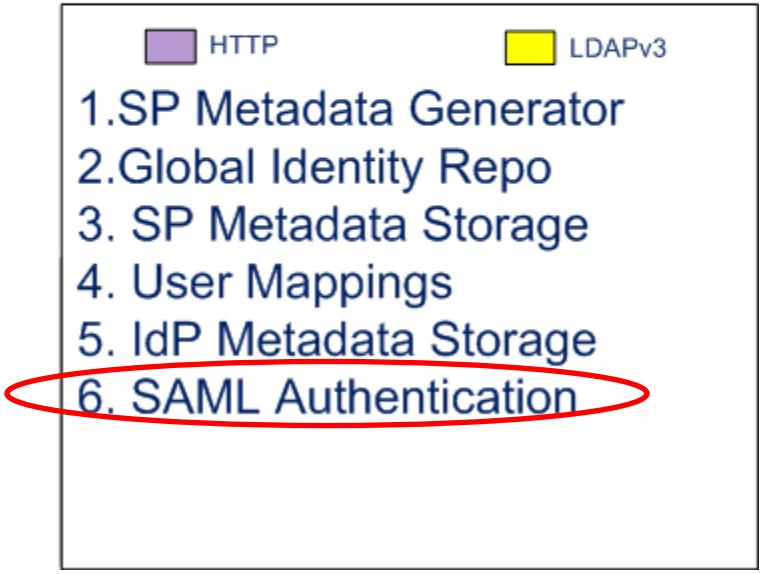
```
     </value>
    </constructor-arg>
    <constructor-arg>
     <value type="int">5000</value>
     </constructor-arg>
     <property name="parserPool" ref="parserPool"/>
   </bean>
  </list>
 </constructor-arg>
</bean>
```

6. Enable Spring SAML Authentication









Enable Spring SAML Security

Add dependencies to pom:

```
<dependency>
  <groupId>org.springframework.security.extensions</groupId>
  <artifactId> spring-security-saml2-core </artifactId>
  <version>1.0.1.RELEASE</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId> spring-security-config </artifactId>
  <version> 3.1.2.RELEASE* </version>
  <scope>compile</scope>
</dependency>
```

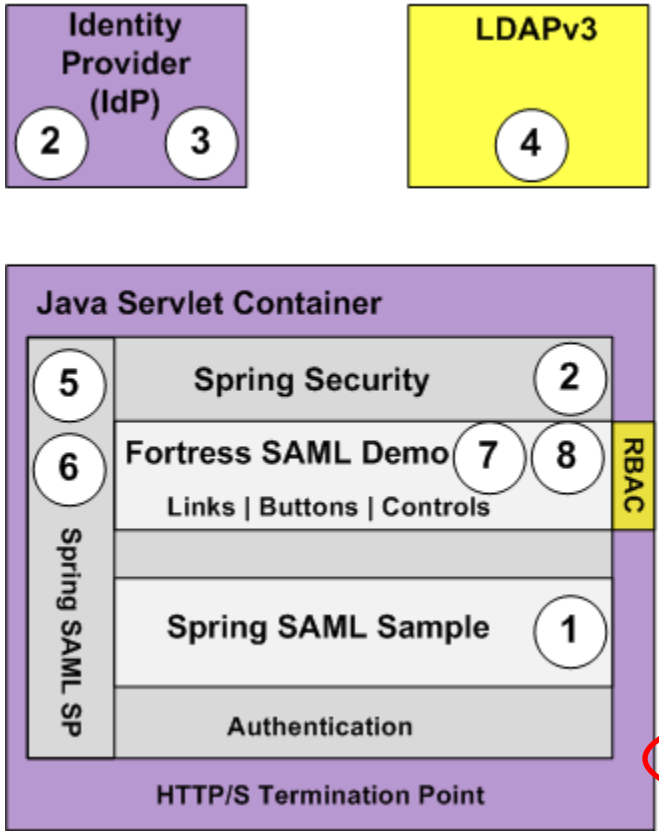
* backlog item

Enable SAML Authentication Filters

In the securityContext.xml

```
<security:http entry-point-ref="samlEntryPoint" use-expressions="false">

  <security:intercept-url pattern="/**" access="IS_AUTHENTICATED_FULLY"/>

  <security:custom-filter before="FIRST" ref="metadataGeneratorFilter"/>
  <security:custom-filter after="BASIC_AUTH_FILTER" ref="samlFilter"/>
</security:http>

<bean id="samlFilter" class="org.springframework.security.web.FilterChainProxy">
    <security:filter-chain-map request-matcher="ant">
    <security:filter-chain pattern="/saml/login/**" filters="samlEntryPoint"/>
    <security:filter-chain pattern="/saml/logout/**" filters="samlLogoutFilter"/>
    <security:filter-chain pattern="/saml/metadata/**" filters="metadataDisplayFilter"/>
    <security:filter-chain pattern="/saml/SSO/**" filters="samlWebSSOProcessingFilter"/>
    <security:filter-chain pattern="/saml/SSOHoK/**" filters="samlWebSSOHoKProcessingFilter"/>
    <security:filter-chain pattern="/saml/SingleLogout/**" filters="samlLogoutProcessingFilter"/>
  </security:filter-chain-map>
</bean>
```

7. Setup RBAC Policy Decision Point



Enable RBAC Policy Decision Point

```
<dependency>
  <groupId>
    org.apache.directory.fortress
  </groupId>
  <artifactId>
      fortress-realm-impl
  </artifactId>
  <version>2.0.0</version>
</dependency>
```

Share ID between Spring & Fortress

Get uid from the SAML assertion, create fortress session.

1. Spring SAML filter creates security principal based on attributes found in the SAML attribute assertion.

2. Web app parses the surName attribute contained in principal:
   `uid=getSurName((SAMLCredential)principal.getCredentials());` <- web app api
3. Web app creates a Fortress session using attribute in the principal:
   `j2eePolicyMgr.createSession( new User( uid ), true );` <- Fortress realm api
4. Web app pushes RBAC session into HTTP session.
   `myAppFw.setSession( ftSess );` <- Web app api

isTrusted (no password req'd)

Parse the ID from SAML Assertion

```
public class SecUtils
{ …
  private static String getSurName(SAMLCredential credential)
  {
    String userId = null;
    for ( org.opensaml.saml2.core.Attribute attr : credential.getAttributes())
    {
      String name = attr.getName();
      if(StringUtils.isEmpty( name ) )
        break;
      else if( name.equals( "LastName" ) )
      {
        String vals[] = credential.getAttributeAsStringArray( attr.getName() );
        userId = vals[0];
        break;
      }
    }
  return userId;
  }
```

Add Secure Web Components

```
public class Page1 extends SamlSampleBasePage
{ …
   add( new FtIndicatingAjaxButton( "Page1", "Add" )
   {
      @Override
      protected void onSubmit( ... )
      {
         // do something here:
      }
   });
}
```

Apache Fortress Saml Demo

- Three Pages
- Each has buttons controlled by RBAC permissions.
- One role per page.
- Users may be assigned to one or more roles.

User to Role	Page One	Page Two	Page Three
Sam*	True	True	True
Sam1	True	False	False
Sam2	False	True	False
Sam3	False	False	True

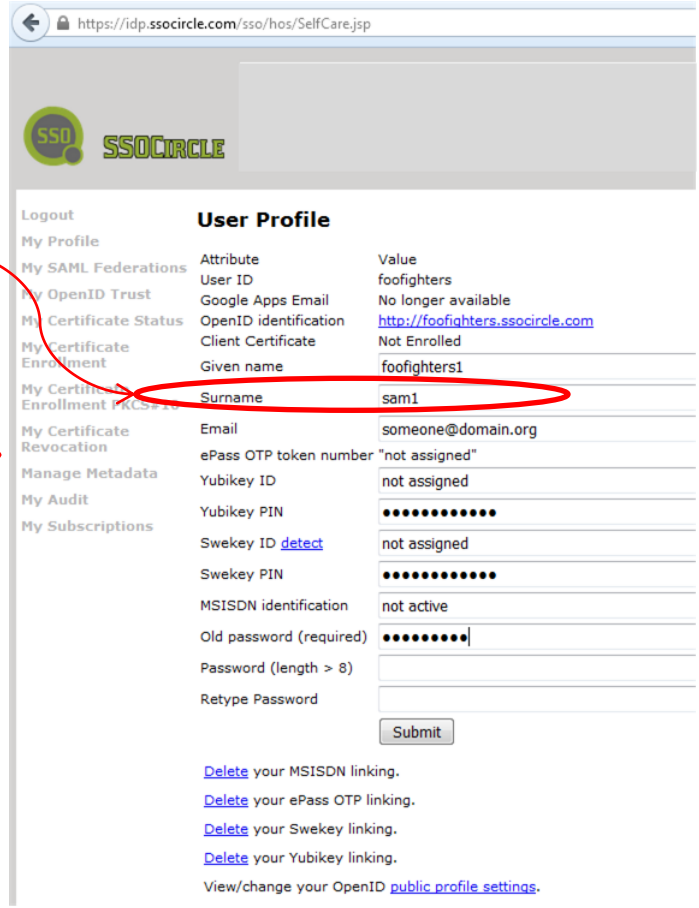
To Change Demo Users

uid=sam1,ou=People,dc=example,dc=com

DN: uid=sam1,ou=People,dc=example,dc=com

Attribute Description	Value
objectClass	***extensibleObject (auxiliary)***
objectClass	***ftMods (structural)***
objectClass	***ftProperties (structural)***
objectClass	***ftUserAttrs (structural)***
objectClass	***inetOrgPerson (structural)***
objectClass	***organizationalPerson (structural)***
objectClass	***person (structural)***
objectClass	***top (abstract)***
cn	**Sam One**
sn	**One**
description	Fortress SAML Demo User 1
displayName	Sam One
ou	org.samlsample.users
uid	sam1
userPassword	SSHA hashed password
ftCstr	*sam1$0$$$$$$$*
ftId	*a59bf210-7101-4bb9-b089-9205d594d108*
ftProps	*init:*
ftRA	*samRole1*

Change Surname field in SSO Circle Profile to use different rbac users.



Apache Fortress SAML Demo

- https://github.com/shawnmckinney/fortress-saml-demo

User to Role	Page One	Page Two	Page Three
Sam*	True	True	True
Sam1	True	False	False
Sam2	False	True	False
Sam3	False	False	True

Closing Thoughts

1. Never allow users more than they need to do their jobs
   - *Principle of Least Privilege*
2. Apply security controls across many layers
   - *Defense in Depth*
3. Use TLS across all remote connections
   - *Confidentiality and Integrity*

Contact Info