The Anatomy of a Secure Web Application Using Java

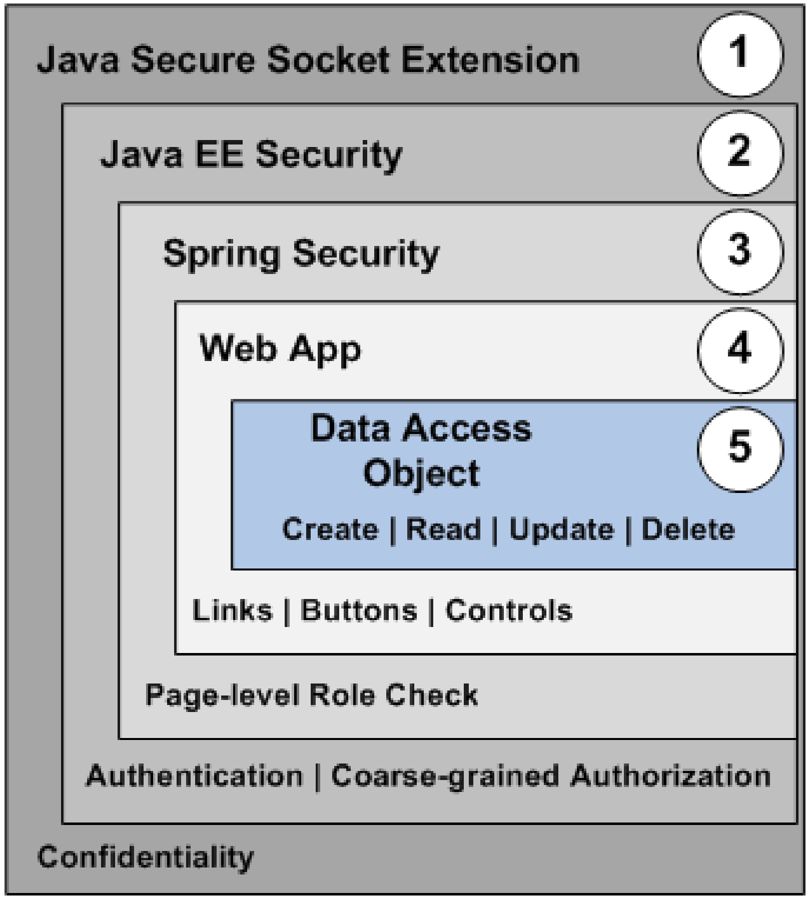
symas

Themes

- Use simple and proven methods for doing security in web apps.
- Use common sense when determining which security measures to take.
- Household analogy to compare security within web systems and home.

The Five Security Layers of Java Web Applications

1. Java Secure Socket Extension
2. Java EE Security
3. Spring Security
4. Web App Framework
5. Database Functions


Java Secure Socket Extension
1
Java EE Security
2
Spring Security
3
Web App
4
Data Access Object
5
Create | Read | Update | Delete
Links | Buttons | Controls
Page-level Role Check
Authentication | Coarse-grained Authorization
Confidentiality

Web to Household Security Analogy

The five security layers relate to everyday concepts:

1. Confidentiality: Privacy in conversation
2. Perimeter: Always lock doors and windows at night and when away. Keeps the bad guys out and the good guys safe.
3. Page Level: Place locks on doors inside the home. For example the media room.
4. App Level: Operation of equipment within a particular room (TV on/off)
5. Data Level: Controls the content of room's equipment (TV channel)

Info on Fortress Demo2 Tutorial

This slide deck describes security functions covered by the Fortress Demo2 tutorial.

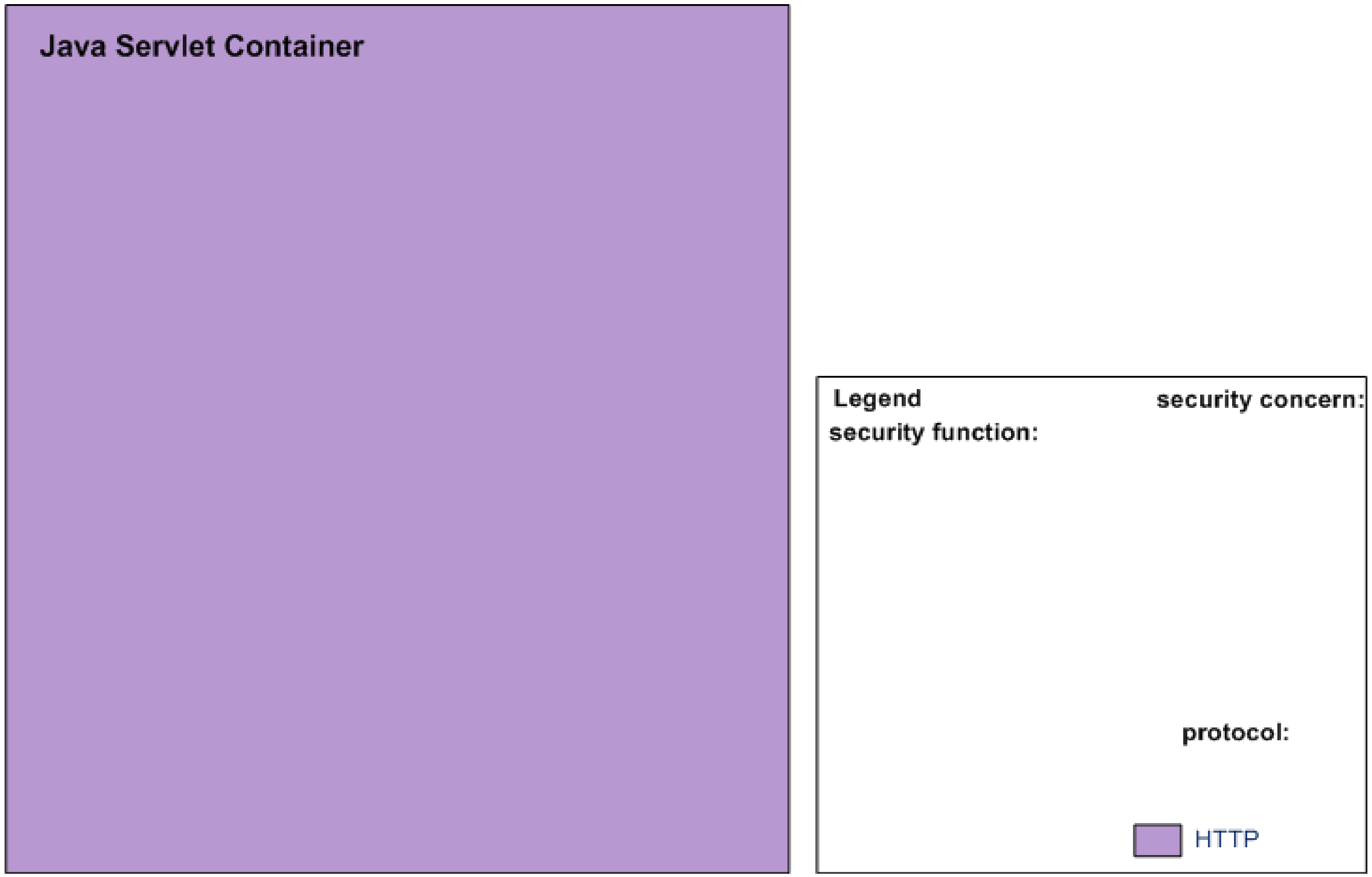
The source code artifacts referenced within these slides link to:

https://github.com/shawnmckinney/fortressdemo2

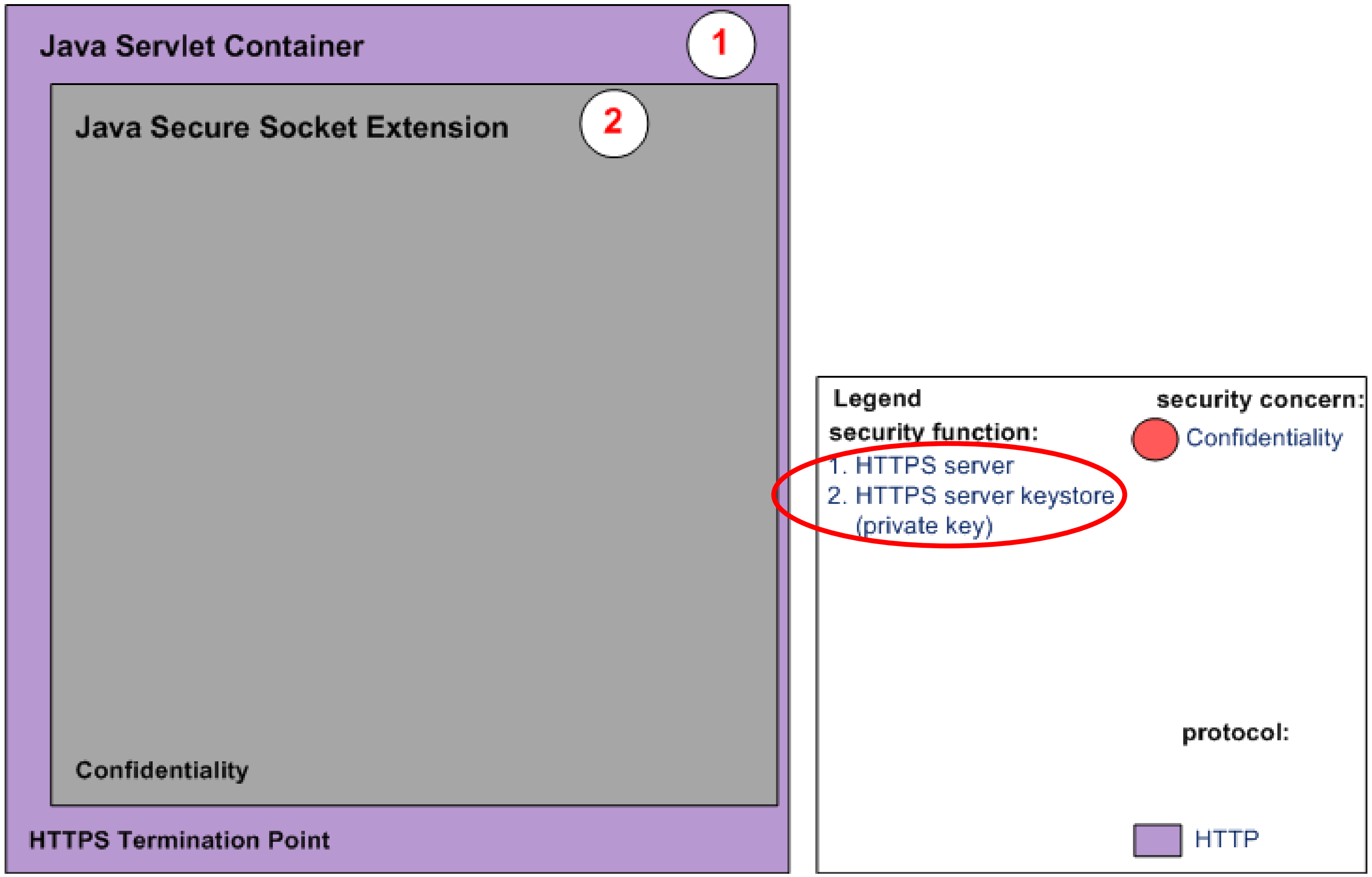
The tutorial's webpage is here:

http://symas.com/kb/demonstrate-end-to-end-security-enforcement-using-open-source/

Start with Tomcat Servlet Container



Enable HTTPS



Enable Tomcat SSL

1. Generate keystore with private key (Steps 1 - 5):
https://symas.com/javadocs/fortressdemo2/doc-files/II-keys.html

2. Add the following to **server.xml**:

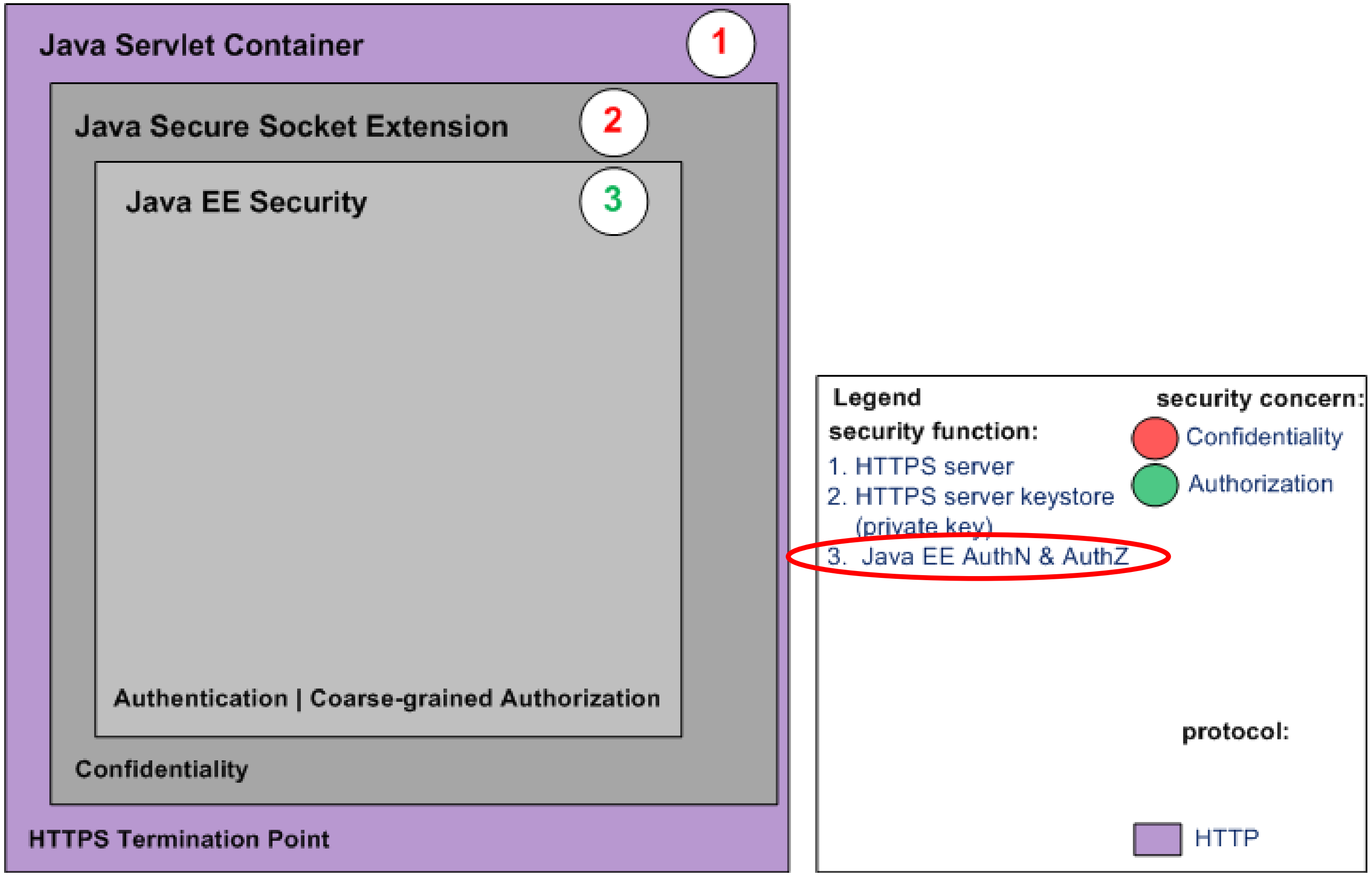
```
<Connector port="8443" maxThreads="200"
   scheme="https" secure="true"
   SSLEnabled="true“
   keystoreFile= “/path/mykeystore”
   keystorePass= “changeit”
   clientAuth="false" sslProtocol="TLS"/>
```

Enable Tomcat SSL

Step 7:

http://symas.com/javadocs/fortressdemo2/doc-files/VI-tomcat.html

Enable Java EE Security

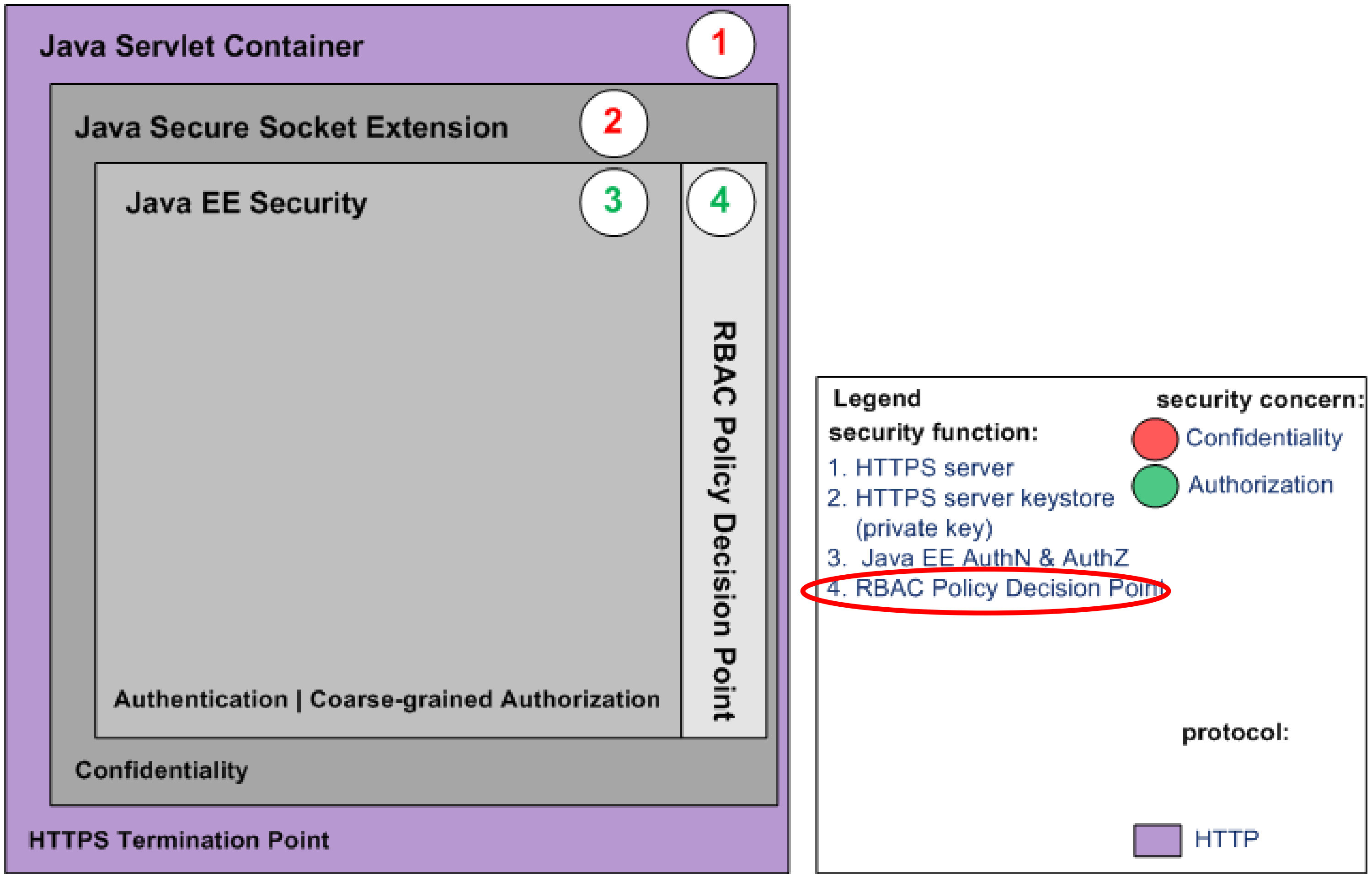


Add to web.xml

```
<security-constraint>
    <display-name>My Security Constraint</display-name>
    <web-resource-collection>
        <web-resource-name>Protected Area</web-resource-name>
        <url-pattern>/secured/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>ROLE_DEMO_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>
        <form-error-page>/login/error.html</form-error-page>
    </form-login-config>
</login-config>
```

coarse-grained authorization (declarative)

Enable Policy Decision Point

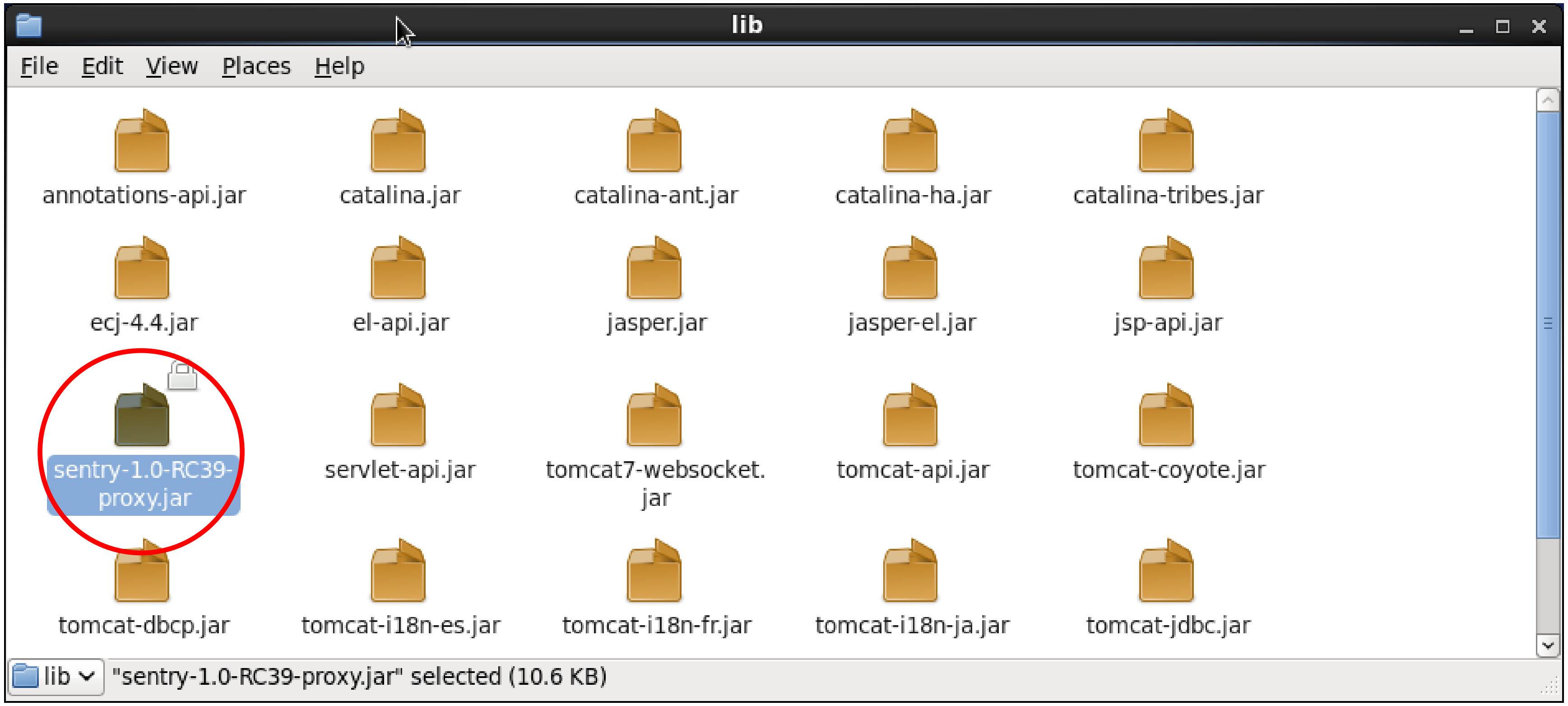


Enable Policy Decision Point

Add context.xml to web project’s META-INF folder:

```
<Context reloadable="true">
    < Realm className=
      "org.openldap.sentry.tomcat.Tc7AccessMgrProxy"
      debug="0"
       resourceName="UserDatabase"
       defaultRoles="ROLE_DEMO2_SUPER_USER,
      DEMO2_ALL_PAGES, ROLE_PAGE1, ROLE_PAGE2,
      ROLE_PAGE3"
       containerType="TomcatContext"
       realmClasspath=""
    />
</Context>
```

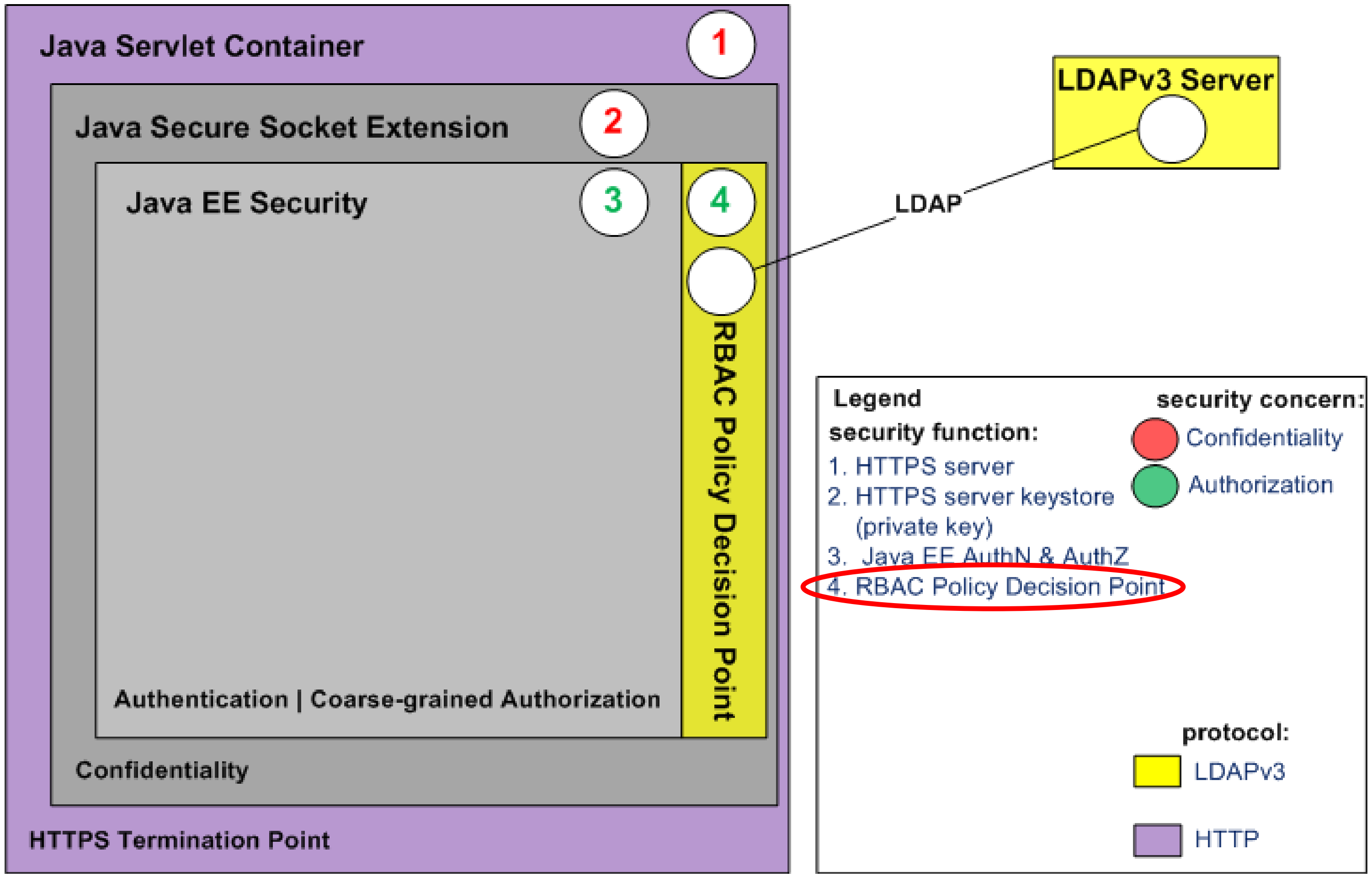
Drop the Sentry proxy jar in Tomcat's system classpath



"sentry-1.0-RC39-proxy.jar" selected (10.6 KB)

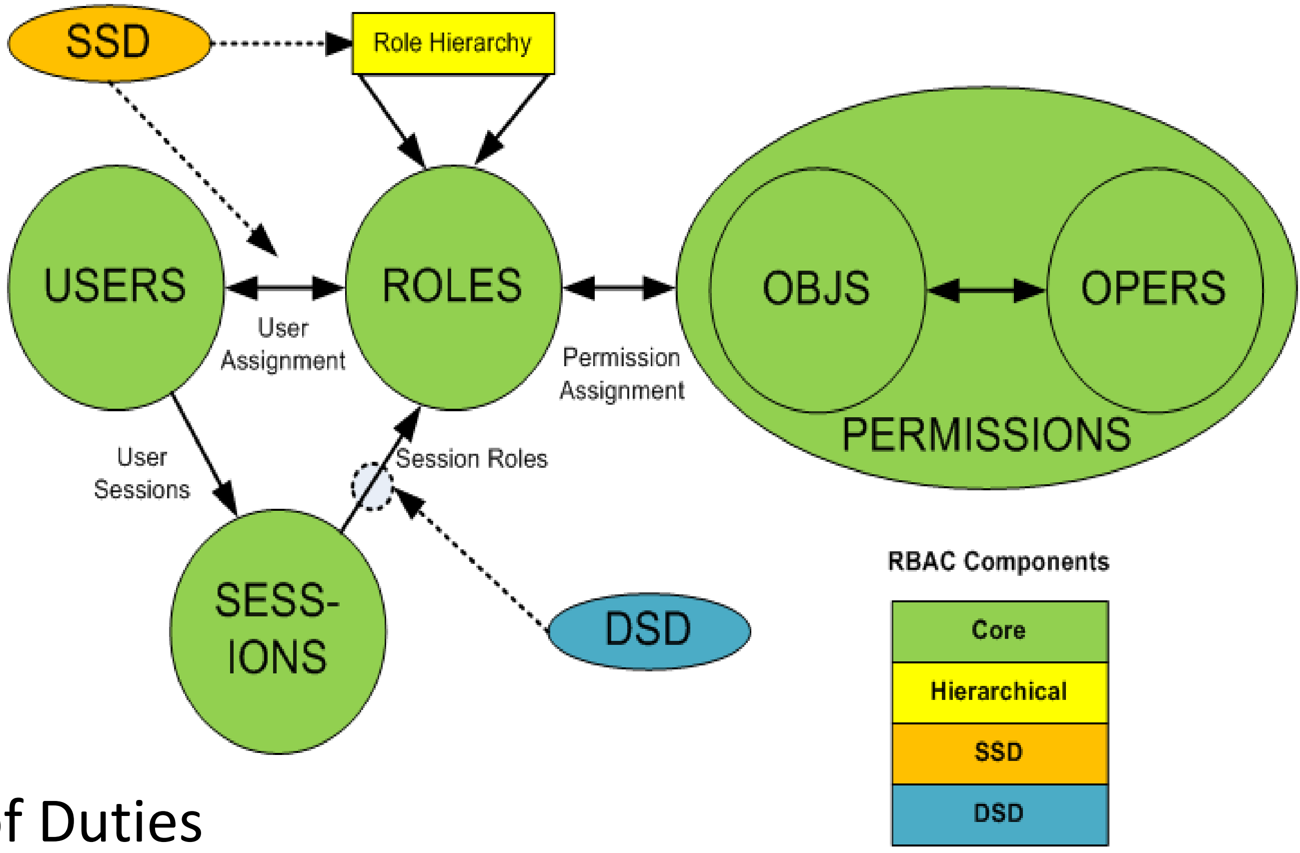
symas

Configure Sentry RBAC PDP



ANSI RBAC INCITS 359

RBAC0:
Users, Roles,
Perms, Sessions
RBAC1:
Hierarchical Roles
RBAC2:
Static Separation
of Duties
RBAC3:
Dynamic Separation of Duties



ANSI RBAC Object Model

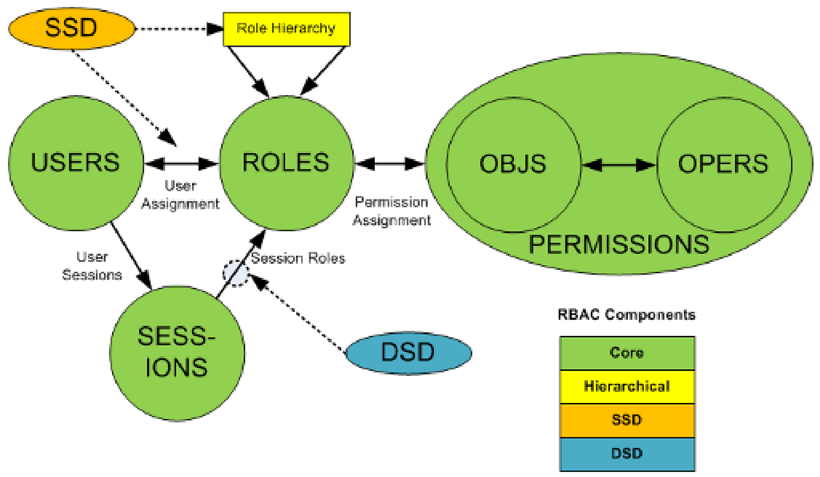
Six basic elements:

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

ANSI RBAC Functional Model

Three standard interfaces:

1. Administrative – CRUD
2. Review – policy interrogation
3. System – policy enforcement



Configure Sentry RBAC PDP

ANSI RBAC Policy Decision Point

http://symas.com/javadocs/fortress/org/openldap/fortress/AccessMgr.html

1. createSession
2. checkAccess
3. sessionPermissions
4. sessionRoles
5. getUser
6. addActiveRole
7. dropActiveRole

Configure Sentry RBAC PDP

Install OpenLDAP Fortress QUICKSTART:

http://symas.com/javadocs/fortressdemo2/doc-files/IV-fortress.html

Configure Sentry RBAC PDP

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

```
<dependency>
   <groupId> org.openldap </groupId>
   <artifactId> sentry </artifactId>
   <version> 1.0-RC39 </version>
</dependency>
```

Configure Sentry RBAC PDP

Add Spring's context file to web app's web.xml file:

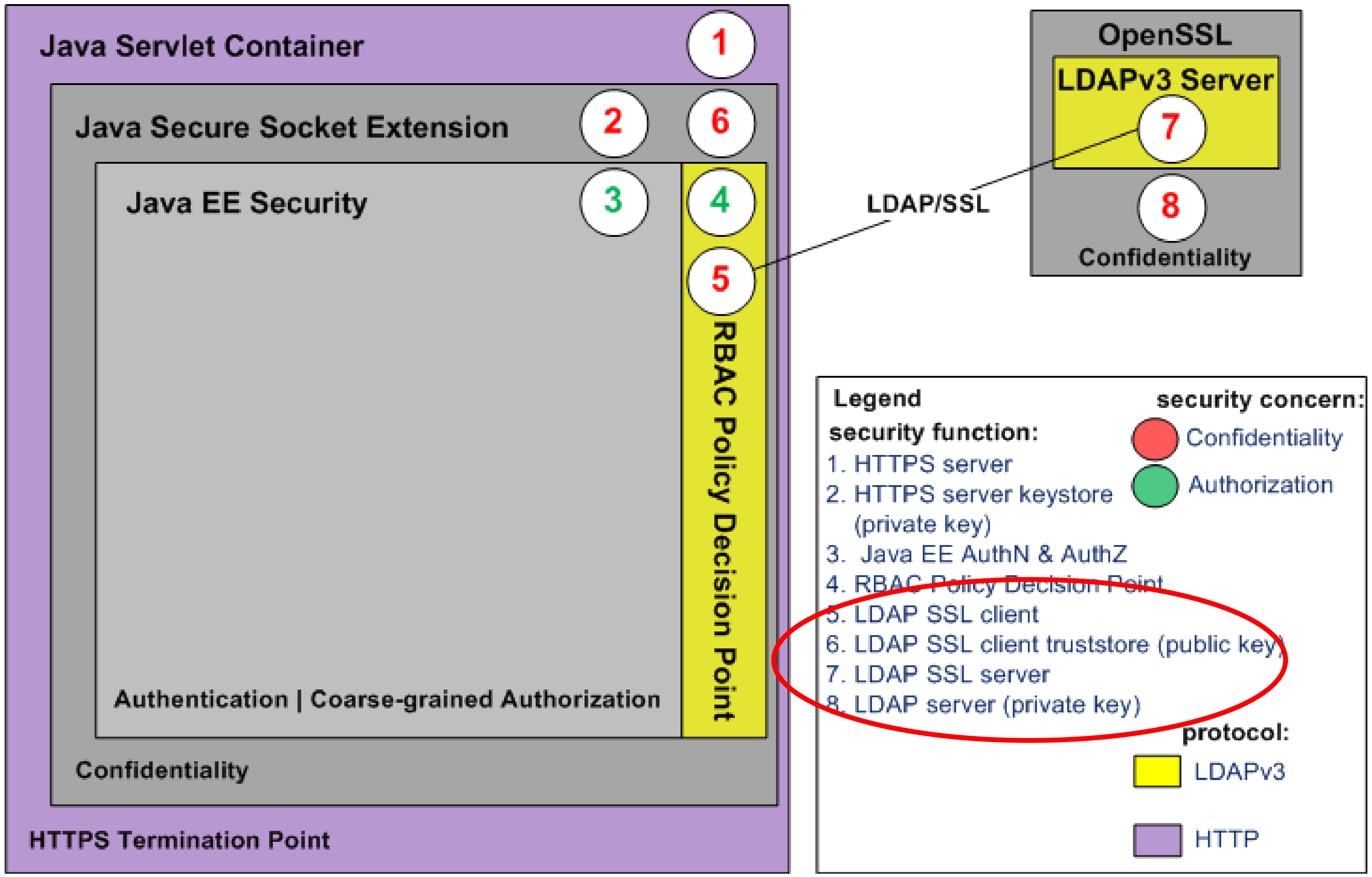
```
<context-param>
  <param-name>
    contextConfigLocation
  </param-name>
  <param-value>
classpath:applicationContext.xml
  </param-value>
</context-param>
```

Configure Sentry RBAC PDP

Enable Sentry RBAC Spring Bean in applicationContext.xml:

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

Enable LDAP SSL



Enable OpenLDAP SSL Server

1. Patch Heartbleed:
http://symas.com/javadocs/fortressdemo2/doc-files/I-opensslheartbleed.html

2. Use OpenSSL to generate keys and certs:
http://symas.com/javadocs/fortressdemo2/doc-files/II-keys.html

3. Add generated artifacts to OpenLDAP slapd.conf:

```
TLSCACertificateFile /path/ca-cert.pem
TLSCertificateFile /path/server-cert.pem
TLSCertificateKeyFile /path/server-key.pem
```

4. Add ldaps to OpenLDAP startup params:

```
slapd … –h "ldaps://hostname:636"
```

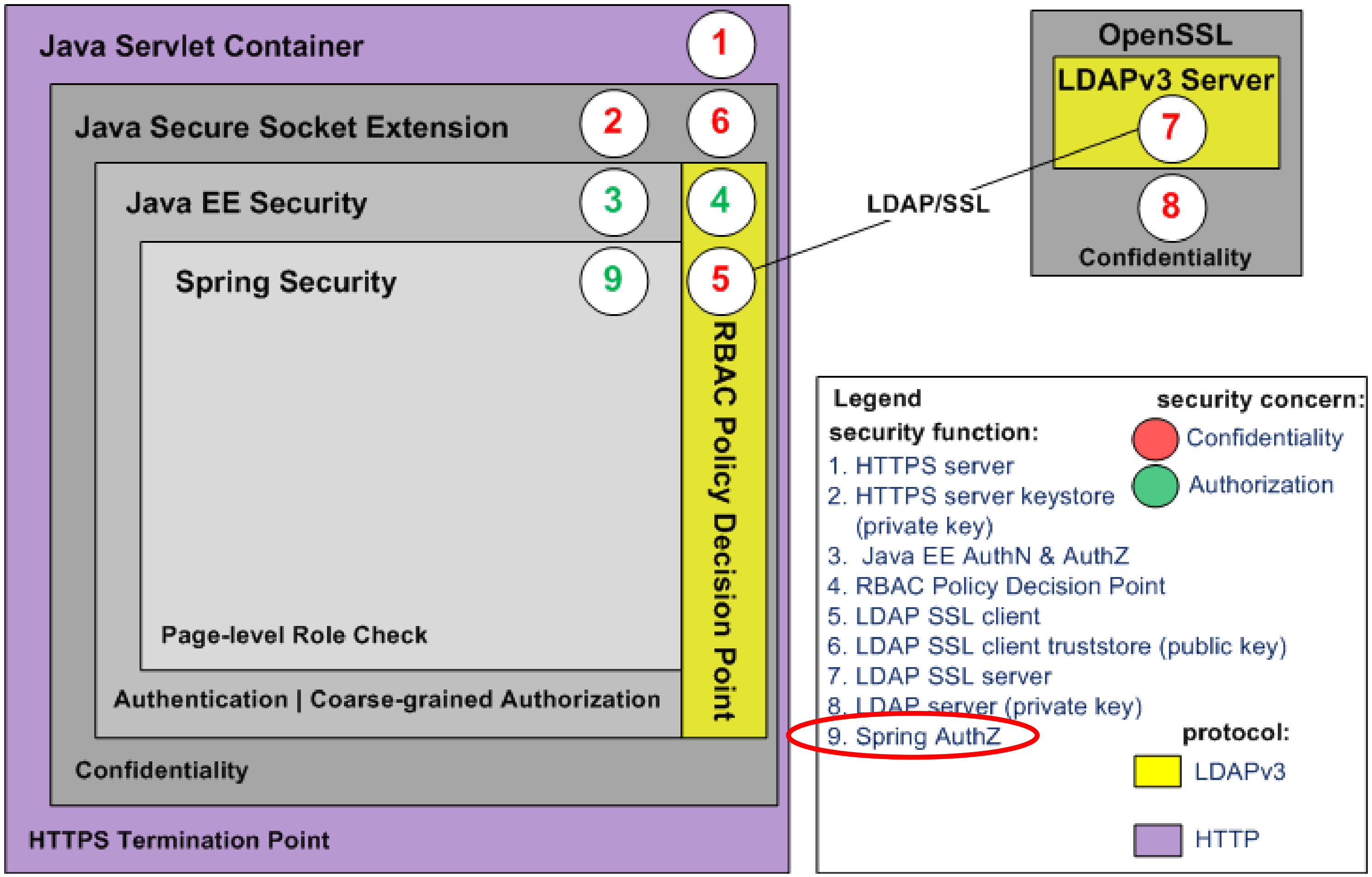
Enable LDAP SSL Client

1. Import public key to java truststore (Step 6):
http://symas.com/javadocs/fortressdemo2/doc-files/II-keys.html

2. Add to fortress.properties of Web application:

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

Enable Spring Security



Enable Spring Security

Add Spring Dependencies to web app's pom.xml:

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId> spring-security-core </artifactId>
  <version>${spring.security.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId> spring-security-config </artifactId>
  <version>${spring.security.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId> spring-security-web </artifactId>
  <version>${spring.security.version}</version>
</dependency>
```

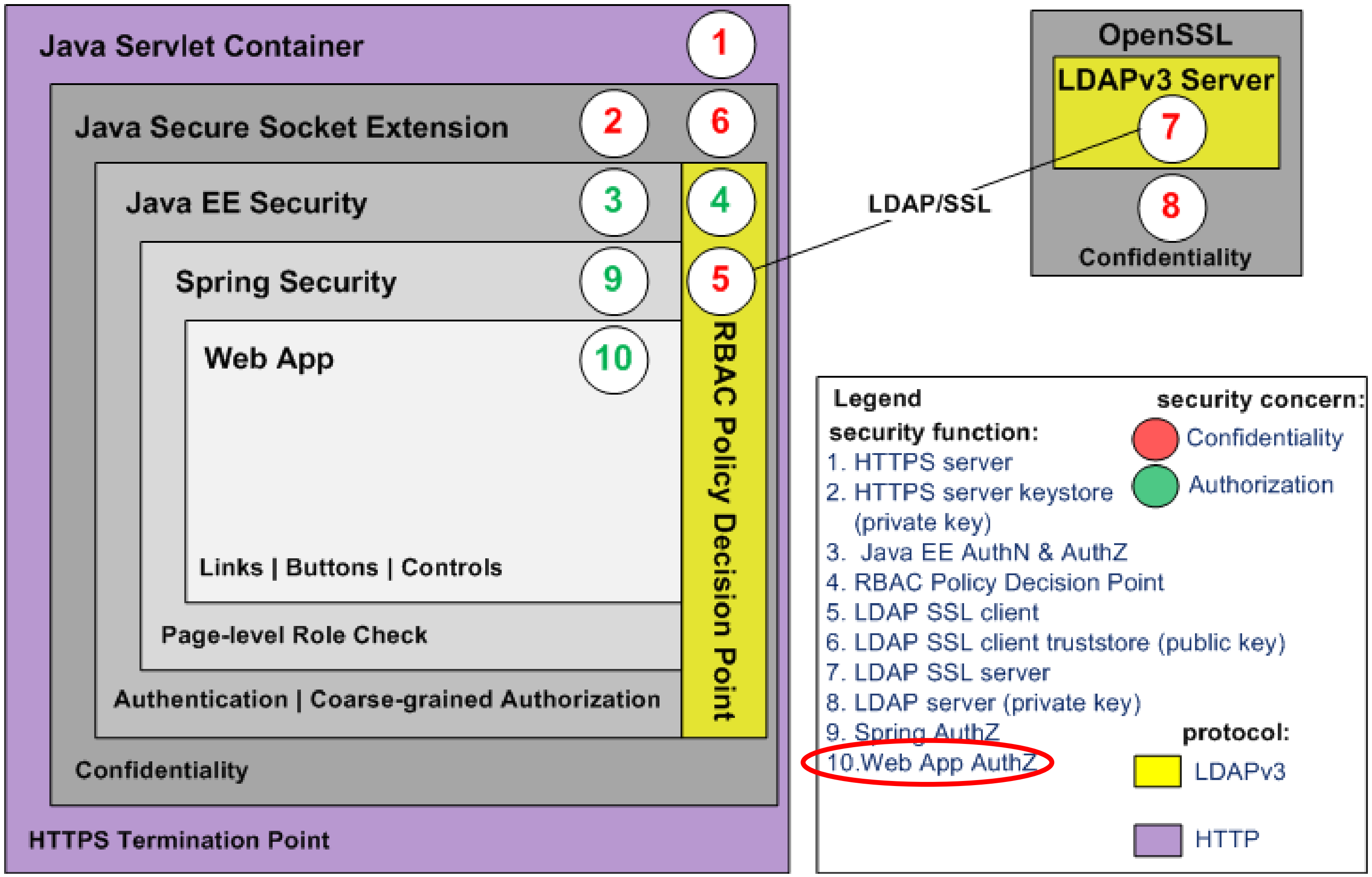
symas

Enable Spring Security

```
<bean id="fsi" class=
   "org.springframework.security.web.access.intercept.FilterSecurityInter
   ceptor">
  <property name="authenticationManager" ref="authenticationManager"/>
  <property name="accessDecisionManager"
   ref="httpRequestAccessDecisionManager"/>
  <property name="securityMetadataSource">
    <sec:filter-invocation-definition-source>
      <sec:intercept-url pattern=
        "/com.mycompany.page1"
        access="ROLE_PAGE1"
      />
</sec:filter-invocation-definition-source>
  </property>
</bean>
```

page-level authorization (declarative)

Add Security Aware Web Framework Components



Legend

security function:
1. HTTPS server
2. HTTPS server keystore (private key)
3. Java EE AuthN & AuthZ
4. RBAC Policy Decision Point
5. LDAP SSL client
6. LDAP SSL client truststore (public key)
7. LDAP SSL server
8. LDAP server (private key)
9. Spring AuthZ
10. Web App AuthZ

security concern:
- 🔴 Confidentiality
- 🟢 Authorization

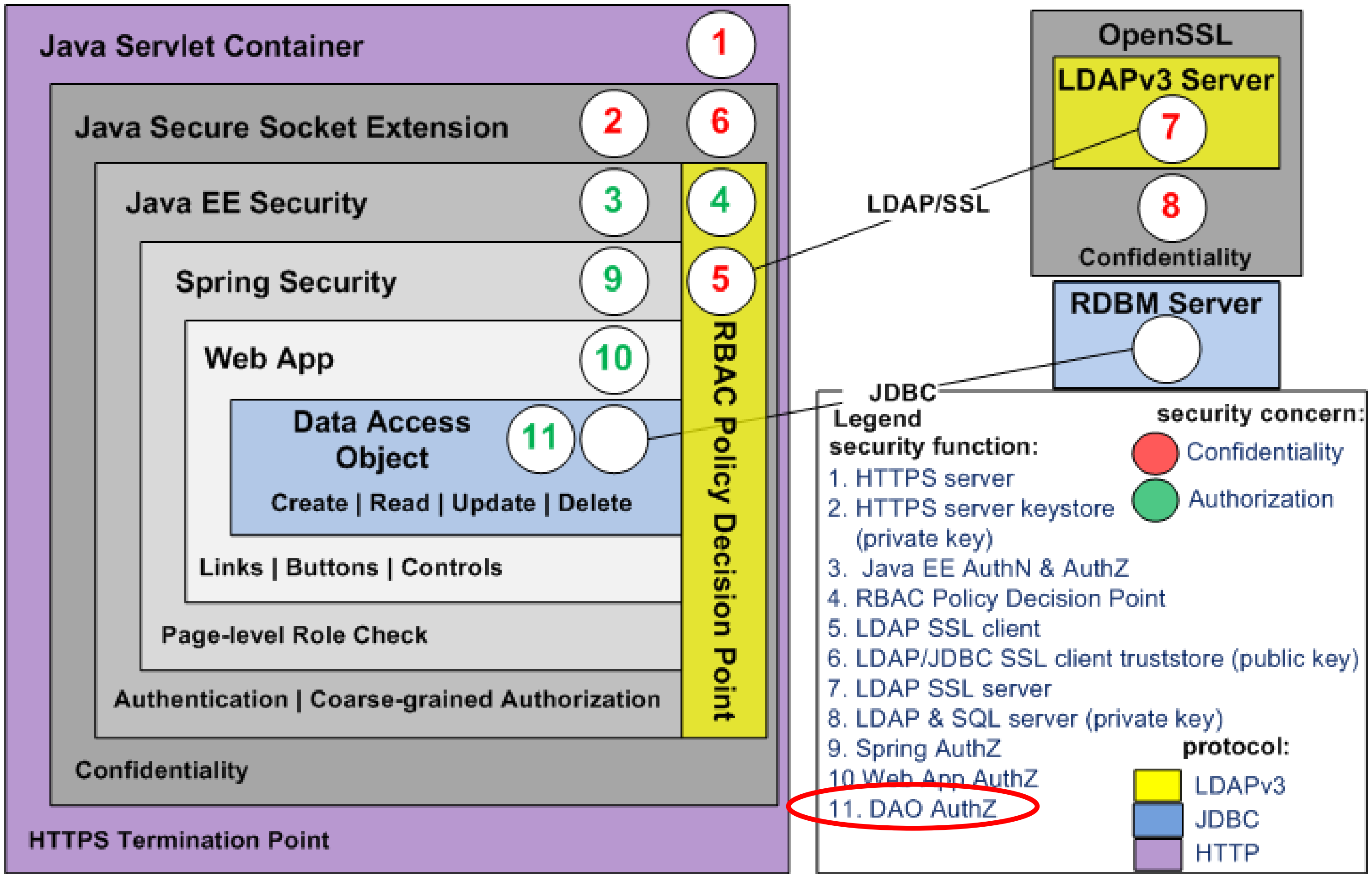
protocol:
- 🟨 LDAPv3
- 🟪 HTTP

Add Security Aware Web Framework Components

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

fine-grained authorization (programmatic)

Add Security Aware DAO components



Legend

security function:
1. HTTPS server
2. HTTPS server keystore (private key)
3. Java EE AuthN & AuthZ
4. RBAC Policy Decision Point
5. LDAP SSL client
6. LDAP/JDBC SSL client truststore (public key)
7. LDAP SSL server
8. LDAP & SQL server (private key)
9. Spring AuthZ
10. Web App AuthZ
11. DAO AuthZ

security concern:

🔴	Confidentiality
🟢	Authorization

protocol:

🟨	LDAPv3
🟦	JDBC
🟪	HTTP

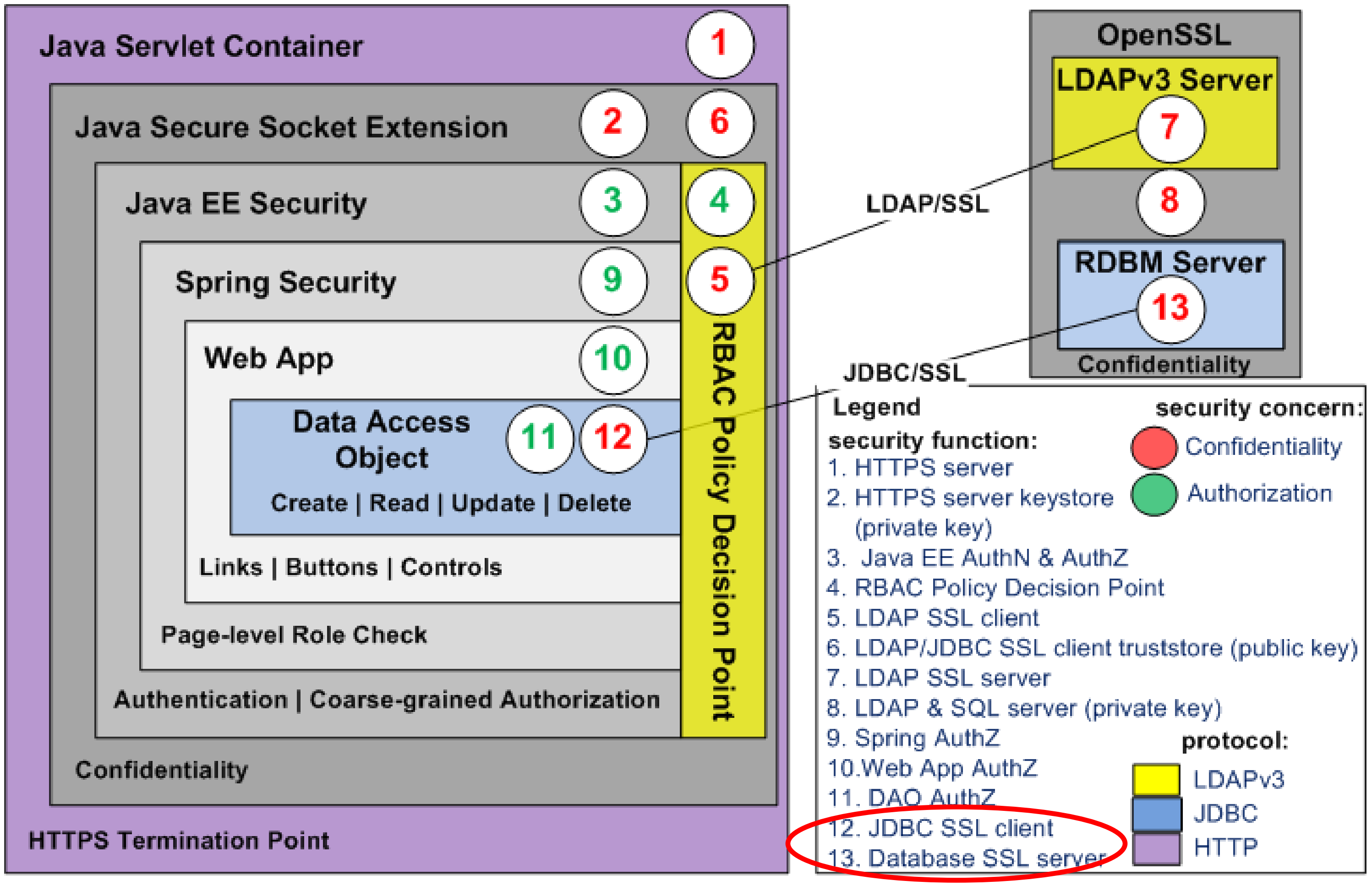
symas

Add Security Aware DAO components

```
public Page1EO updatePage1( Page1EO entity )
{
   ...
 if(checkAccess("Page1","Update",entity.getCust()))
{
   // Call DAO.update method...
}
else
  throw new RuntimeException("Unauthorized");
...
return entity;
}
```

fine-grained authorization (programmatic)

Enable DB SSL



symas

Enable MySQL SSL Server

Add to MySQL my.cnf file:

1. Instruct listener to use host name in certificate:

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

2. Add generated OpenSSL artifacts:

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

Enable MySQL SSL Server

Step 7:

http://symas.com/javadocs/fortressdemo2/doc-files/V-mysql.html

Enable MySQL SSL Client

Add to fortress.properties of Web application:

```
# Sets trust.store params as
  System.property to be used by JDBC
  driver:
trust.store.set.prop=true

# 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
```

Demo

- https://symas.com/javadocs/fortressdemo2/
- https://github.com/shawnmckinney/fortressdemo2
- https://symas.com/javadocs/fortressdemo2/doc-files/VIII-demo.html

Thank You

symas