Java is a trademark of Sun Microsystems, Inc.

Securing Web and SOA with Apache Axis, WSS4J, Spring, and OpenLDAP

Shawn McKinney

Mike Scheuter

Fidelity National Information Services

Marty Heyman

SYMAS

“I had to keep guessing at the channel; I had to discern, mostly by inspiration, the signs of hidden banks; I watched for sunken stones; When you have to attend to things of that sort, to the mere incidents of the surface, the reality—the reality, I tell you—fades. The inner truth is hidden.”

Joseph Conrad, *Heart of Darkness*

Goal

Provide you with a blueprint for building a security solution for your Web and SOA environment.

- Leverage free and open software
- Provide security architecture solutions
- Patterns for custom identity and access management

Agenda

3 Steps to free your apps of AAA solutions

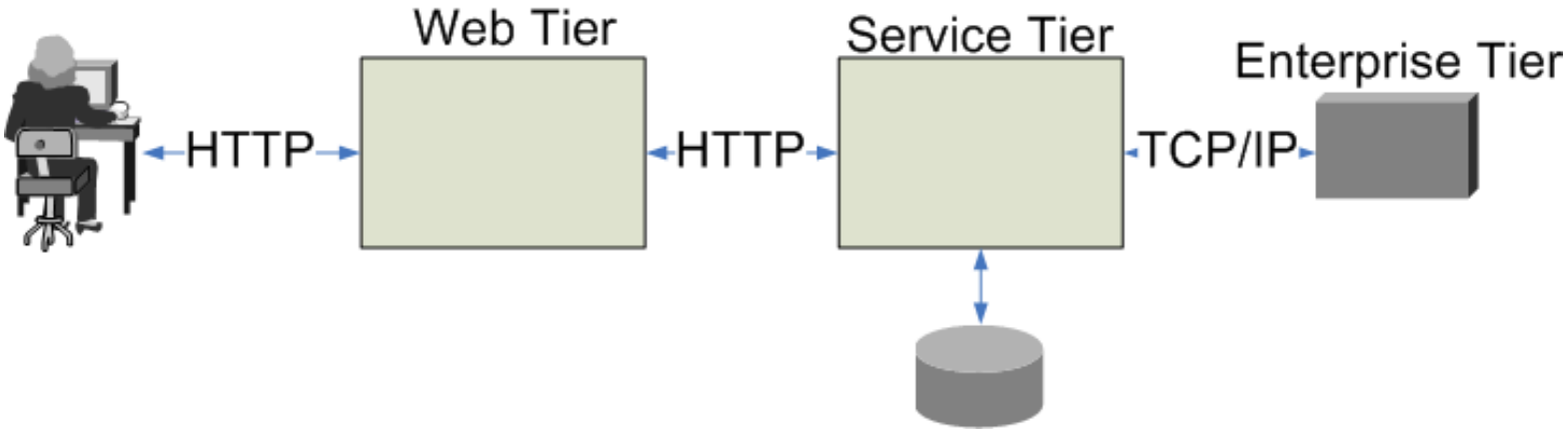
1. Describe isolation techniques
2. Present directions for creating your own security provider
3. Show how to utilize open-source security repository

The Stage: Concerns for our situation

1. Had to be pluggable
2. Standard where possible
3. Facilitate the separation of concerns
4. Fast
5. Conservative
6. Cost effective
7. Proven
8. Secure

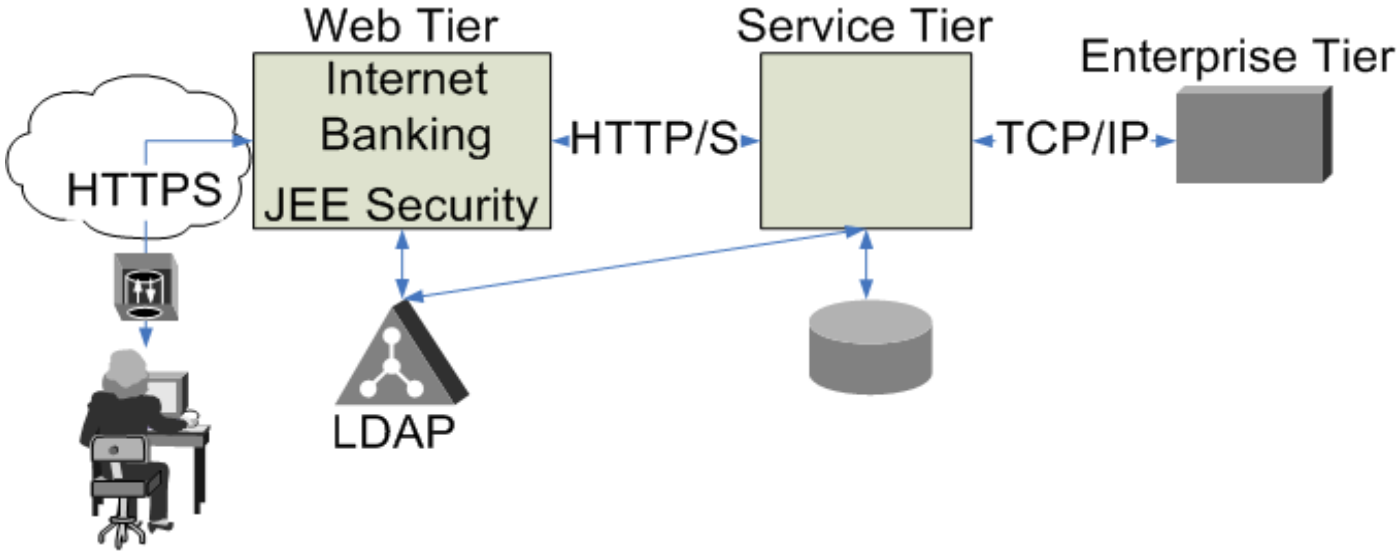
Big Picture

- N-Tier Solution
  - E-Commerce Web application
    - Licensed Client



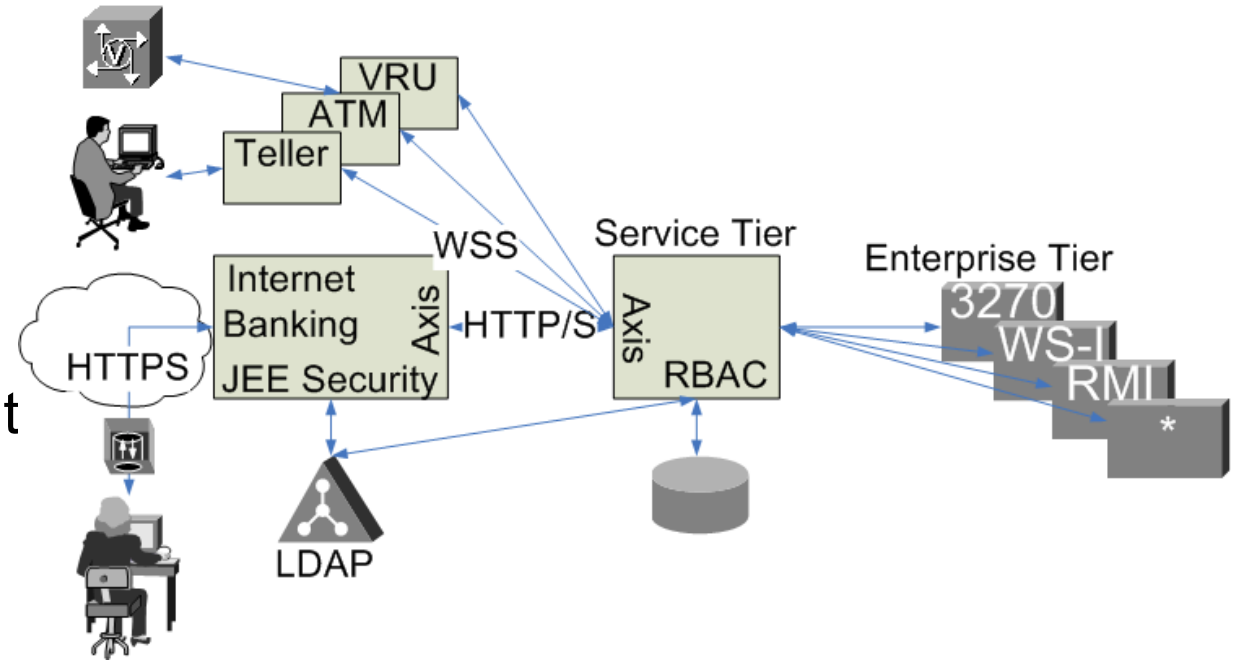
Big Picture

- N-Tier Solution
  - E-Commerce Web application
    - Licensed Client
    - Many Users
    - Internet



Big Picture

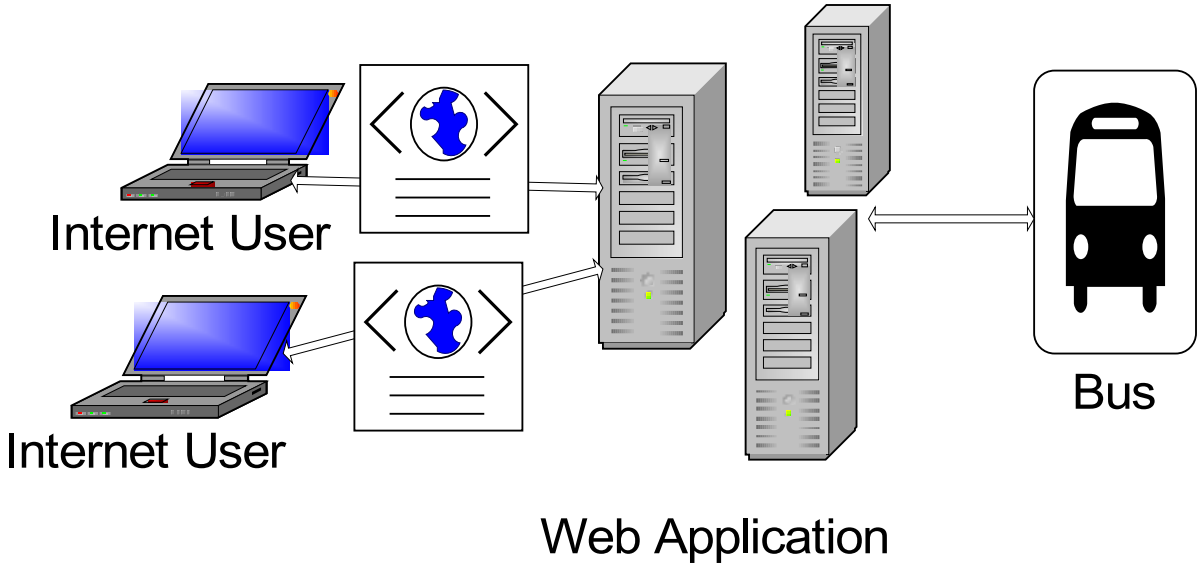
- N-Tier Solution
  - E-Commerce
    - Licensed Client
    - Many Users
    - Internet



    - Trusted connection to Service tier
    - Document-based message architecture
    - SOAP/WSS
    - Integration brokering, middle to enterprise
    - Credential mapping to Enterprise Tier

Web Tier

Authentication, Authorization, Message Security



- Steps for Web Tier Security

1. Use J2EE
2. Use WS-Security/SSL

Web Tier
Step 1 - Use J2EE Security

- Why
  - Separation of concerns
  - Standards/Integration
  - SSO
  - Container does the heavy lifting
  - Bullet-proof
- How
  - Enable container security for your application
  - Implement container's custom security realm SPI
  - Or - use the JSR-115/JSR-196 SPI

Web Tier
Step 2 - Use WS-Security & SSL

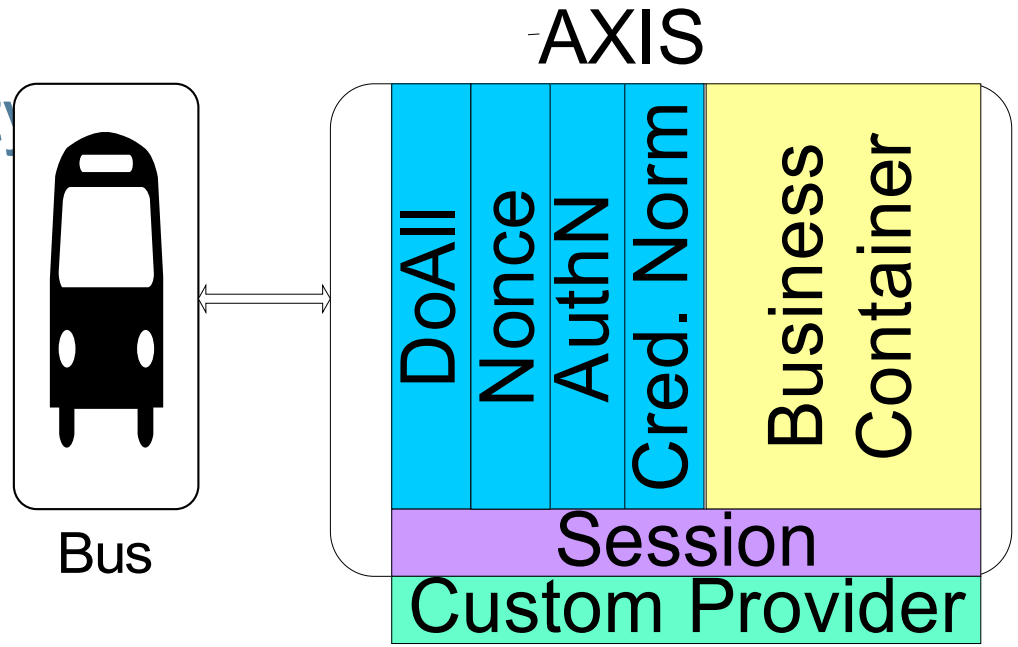
- Where
  - WSS4J DoAllSender in Axis
- Why
  - Standard and secure means to share credentials with service tier
  - Message confidentiality/integrity
  - Axis does heavy lifting
- How
  - Use Rampart/WSS4J for Username Token creation, encryption & signature
  - Enable Axis HTTPS for payload encryption

SOAP Gateway

Web Services - Axis

SOAP Gateway

Message-based Security



- Concerns
  - Message Security (credential passing, confidentiality, integrity)
  - Authentication
  - Session Management
  - Normalize inbound credential

SOAP Gateway

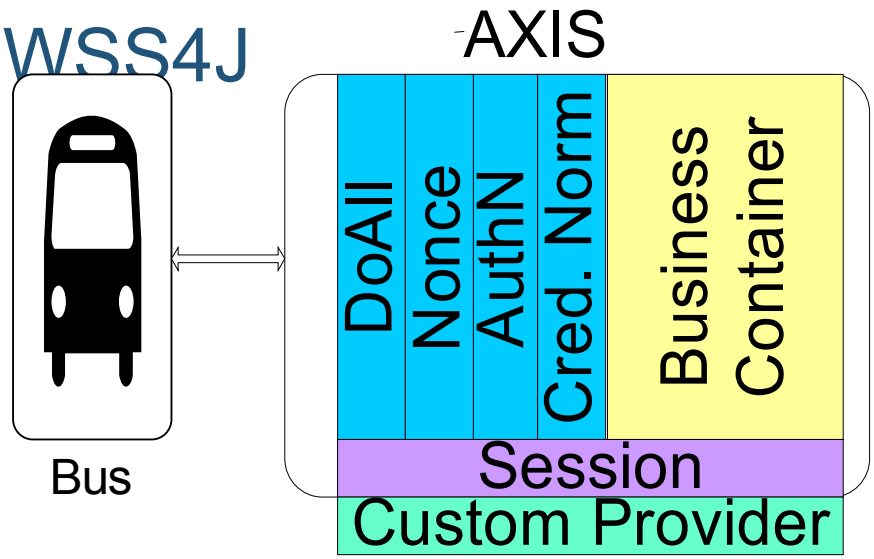
4 Steps

1. Enable Axis, Rampart & WSS4J message security
2. Implement Custom Nonce Handler
3. Implement Custom Authentication Handler
4. Implement Custom Credential Mapping

SOAP Gateway

Step1 Use Axis, Rampart & WSS4J

- Why
  - Out of the box message security
  - Flexible
  - Declarative
  - Support WS- Security
- How
  - Enable Rampart module



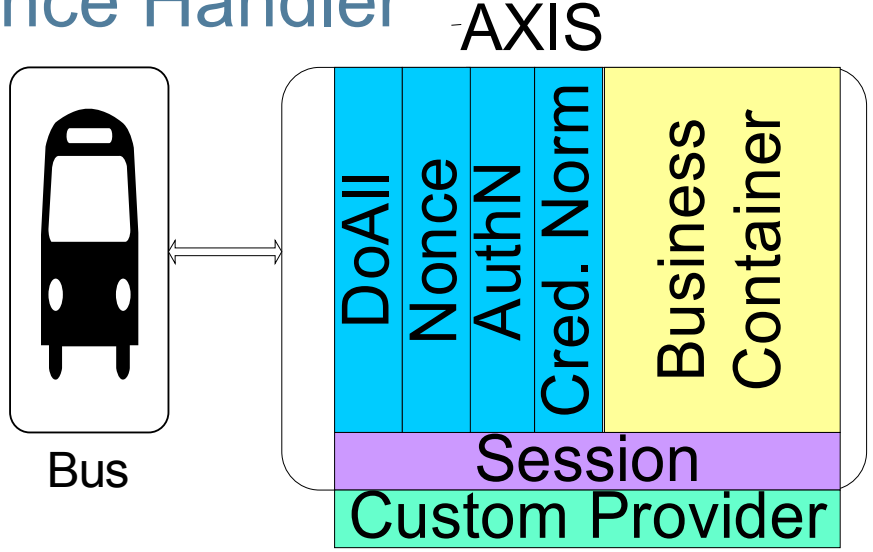
SOAP Gateway

rampart-1.4.mar

```
<InFlow>
   <handler name="PolicyBasedSecurityInHandler"
   class="org.apache.rampart.handler.RampartReceiver">
         <order phase="Security" phaseFirst="true"/>
   </handler>
   <handler name="SecurityInHandler"
   class="org.apache.rampart.handler.WSDoAllReceiver">
         <order phase="Security"/>
   </handler>
   <handler name="PostDispatchVerificationHandler"
   class="org.apache.rampart.handler.PostDispatchVerificationHan
   dler">
         <order phase="Dispatch" phaseLast="true"/>
   </handler>
</InFlow>
```

SOAP Gateway

Step 2 - Write a Custom Nonce Handler

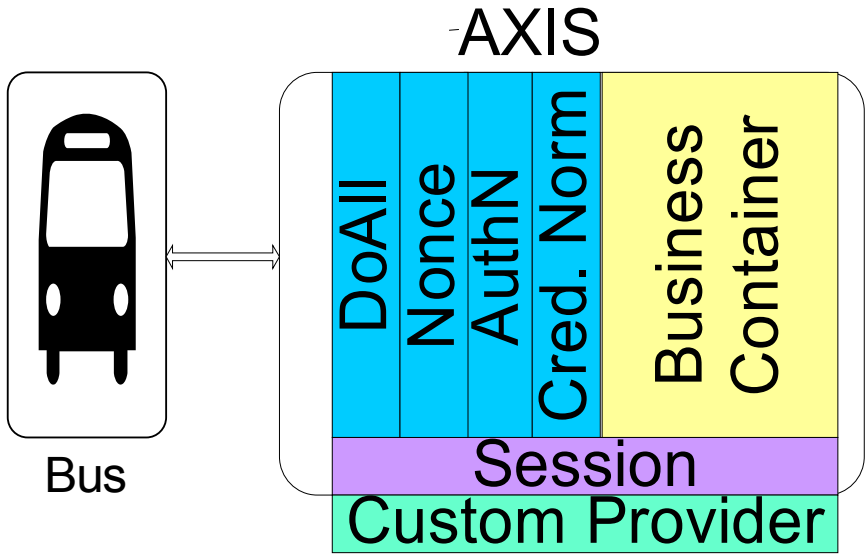


- Why
  - Prevent re-play attack
  - Not provided by toolkit
- How
  - Implement Axis Handler
    - Check for duplicate nonce values
    - Use distributed cache like JCS
    - Only as far back as TTL of UNT
  - Configure Handler Global/Message Inflow

SOAP Gateway

Step 3 - Write a Custom Authentication Handler

- Why
  - Validate Credentials
  - Authenticate Early (DOS)
  - Abstract solution provider
  - Establish RBAC Session
- How
  - Implement Custom Handler
  - Use Provider interface to map to external IAM system
  - Employ caching for RBAC session (JCS)



SOAP Gateway
More on Session Cache

- Why
  - Necessary when session token not in use
  - Holds Roles and other security attributes for the user
  - Performance – don't reconstruct the session
- How
  - Utilize JCS for cache
  - userId/pswd hash for key
    - Password change
  - Time-to-live = Time-to-live for web session

SOAP Gateway
Axis, WSS4J & Custom AuthN

- Choose from one of 10 possible combinations:

Option	A - SSL Mode	B – UNT Type	C – AuthN Mode
1	Off	UNT, TS, Nonce	Trusted
2	On	UNT, Encrypt, TS, Nonce	Shared Secret
3	On/Mutual AuthN	UNT, Signature, TS, Nonce	Un-trusted
4		UNT, Encrypt, Signature, TS, Nonce	

User Profile	Channel Profiles
External	A1, B2, C2
Internal	A1, B3, C1
VRU	A2, B1, C2
Proxy	A3, B1, C2
ATM	A3, B3, C3
…	

SOAP Gateway
WS-Policy

- Scenario: UsernameToken, Encrypt, Timestamp
- Native Rampart enablement

From service.xml:

```
<module ref="rampart" />
<parameter name="InflowSecurity">
    <action>
    <items>UsernameToken Encrypt Timestamp</items>
<passwordCallbackClass>com...LocalPwCbHandler</pass...>
    </action>
</parameter>
```

SOAP Gateway

WS-SecurityPolicy

- Scenario: UsernameToken, Encrypt, Timestamp
- WS-SecurityPolicy advertisement

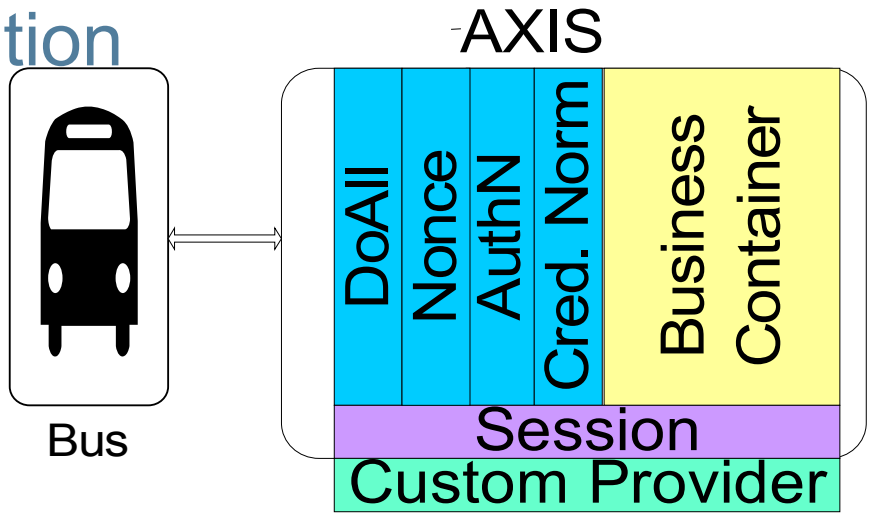
```
<wsp:Policy>
    <sp:SupportingToken>
        <wsp:Policy>
        <sp:UserNameToken>
        <wsp:Policy><sp:HashPassword></wsp:Policy>
        <sp:UserNameToken>
        </wsp:Policy>
    </sp:SupportingToken>
    <sp:EncryptedParts ...>
    <sp:Header Name="Security" Namespace=http://schemas...secext
    </sp:EncryptedParts>
    ...
```

SOAP Gateway
WS-SecurityPolicy

- Standard means of policy advertisement
- Expressive model for defining security policy
- You will want to use tooling to define Security Policy

SOAP Gateway

Step 4 - Credential Normalization



- Why
  - Isolate external credentials from services – necessary abstraction layer from external IAM.
- How
  - Pick a credential representation
    - We use Java Security Principal
  - Write custom credential normalization handler

SOAP Gateway
Example

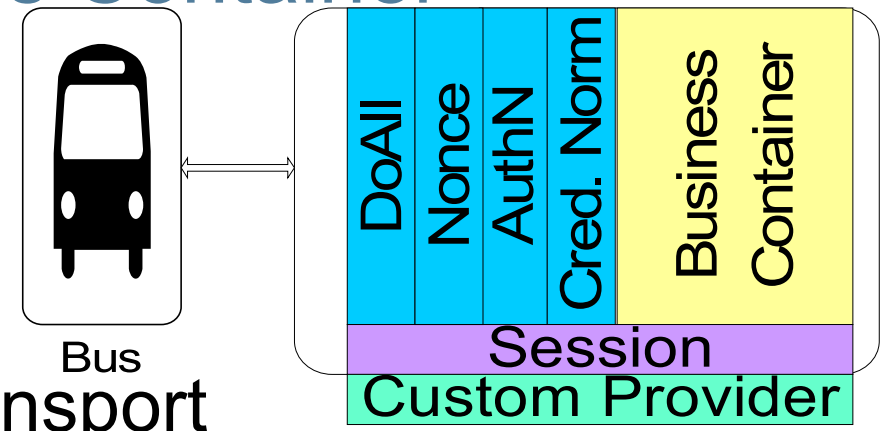
```
<module name="XyzSecurity" class="com...XyzSecurity">
<InFlow>
    <handler name="NonceHandler" class="com.....NonceHandler">
    <order phase="AppSec" phaseFirst="true"/>
    </handler>
    <handler name="AuthNHandler" class="com...XyzAuthNHandler">
    <order phase="AppSec" />
    <parameter name="trusted">true</parameter>
    <parameter
       name="providerClassName">com...XyzSecurityProvider</parameter>
    </handler>
    <handler name="CredNormHandler" class="com....CredNormalize">
    <order phase="AppSec" phaseLast="true"/>
    </handler>
</InFlow>
```

SOAP Gateway
Lessons learned

- Don’t roll your own message based security protocol or security token.
- Focus on key management.
  - Understand how you will swap keys in future and document process.
- Know thy SOAP toolkit.
  - Axis1 threading issues.
  - Maintain pool of Axis stubs on client.
  - Submit bug fixes to Axis

Soap Gateway

Transitioning To Business Container

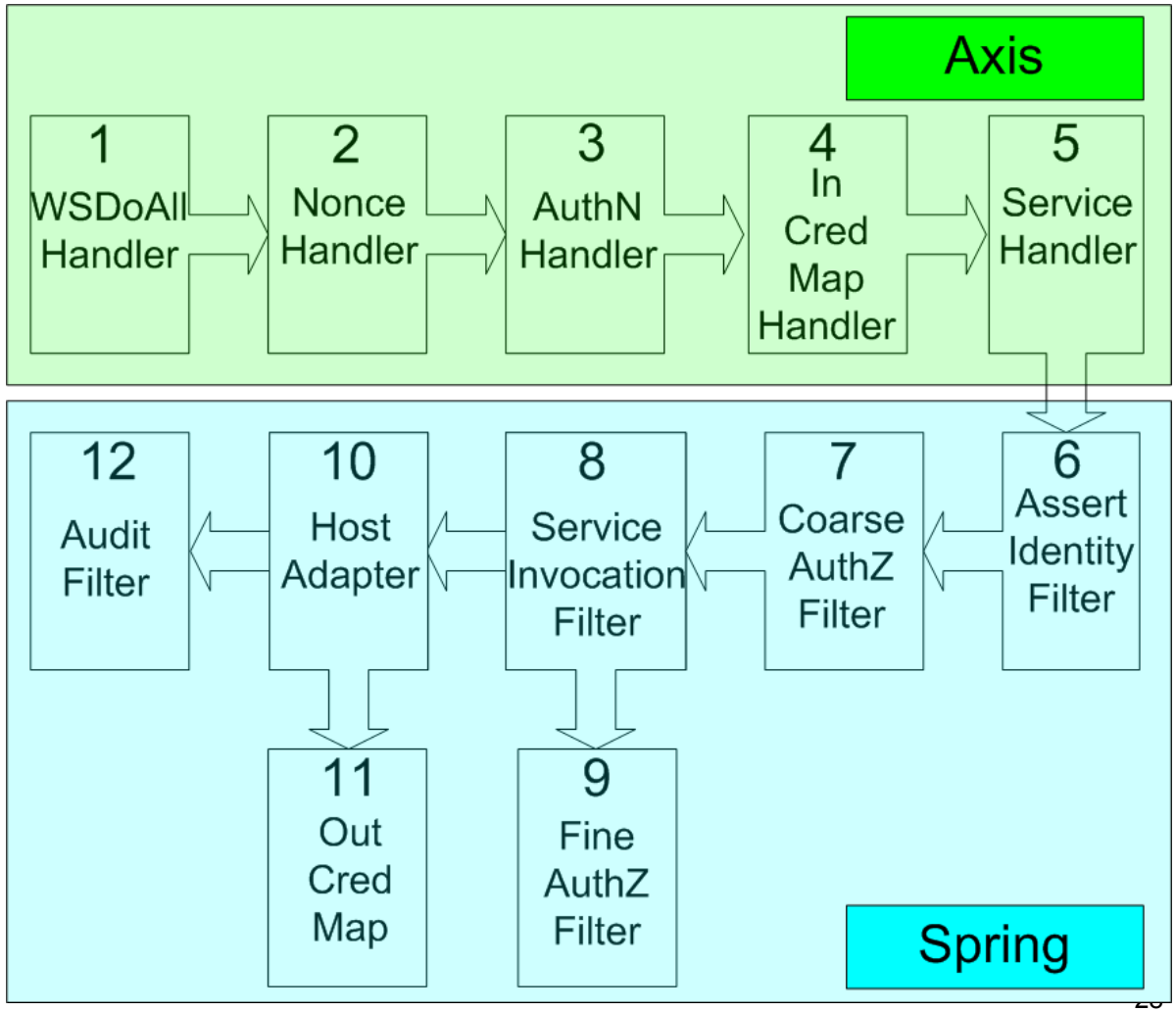


- Isolate application from transport
  - Need to deploy in other environments
- Leverage Spring (DI) Framework
  - Adaptable to new requirements
  - Application framework configuration
  - Aids in testing
- Use several integration patterns in message flow

Business Container

Message Architecture Overview

- Processing transitions From Axis to Business Container
- Business flow represented as filters and routers



Business Container

4 Steps

1. Assert Identity Filter
2. Authorization Filters
3. Audit Filters
4. Credential Mapping (Integration)

Business Container

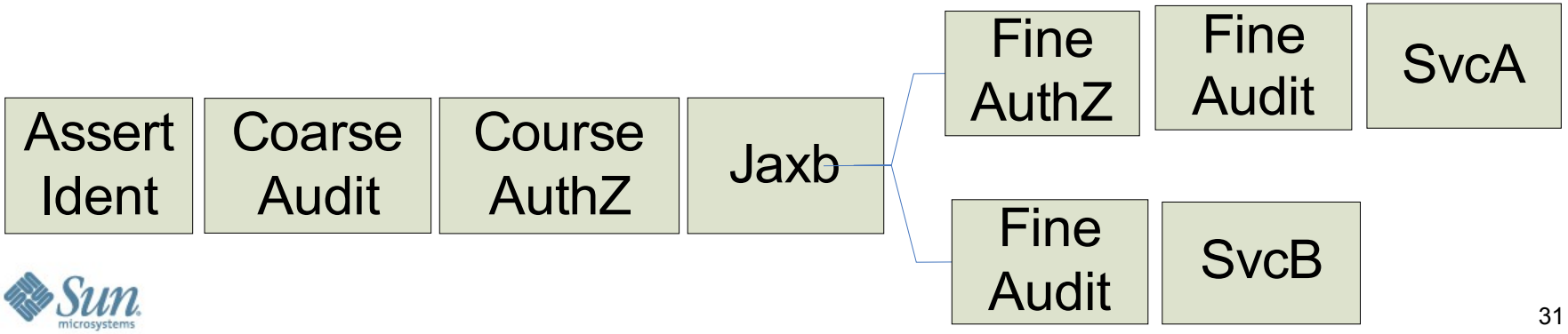
Security Architecture Goals Revisited

- Use declarative policy enforcement
  - Separation of concerns
  - Less chance of programmer mistakes
- Use infrastructure for security
  - Keep security API separate from business logic

Business Container
Message Architecture Overview

- Service fulfillment process like Servlet filters
  - Filters have specific responsibility
  - Message flow is configurable
  - Added Message Routing
  - You could use Spring integration, Camel, Mule ESB

- Filters and dependent classes hosted in Spring



Business Container

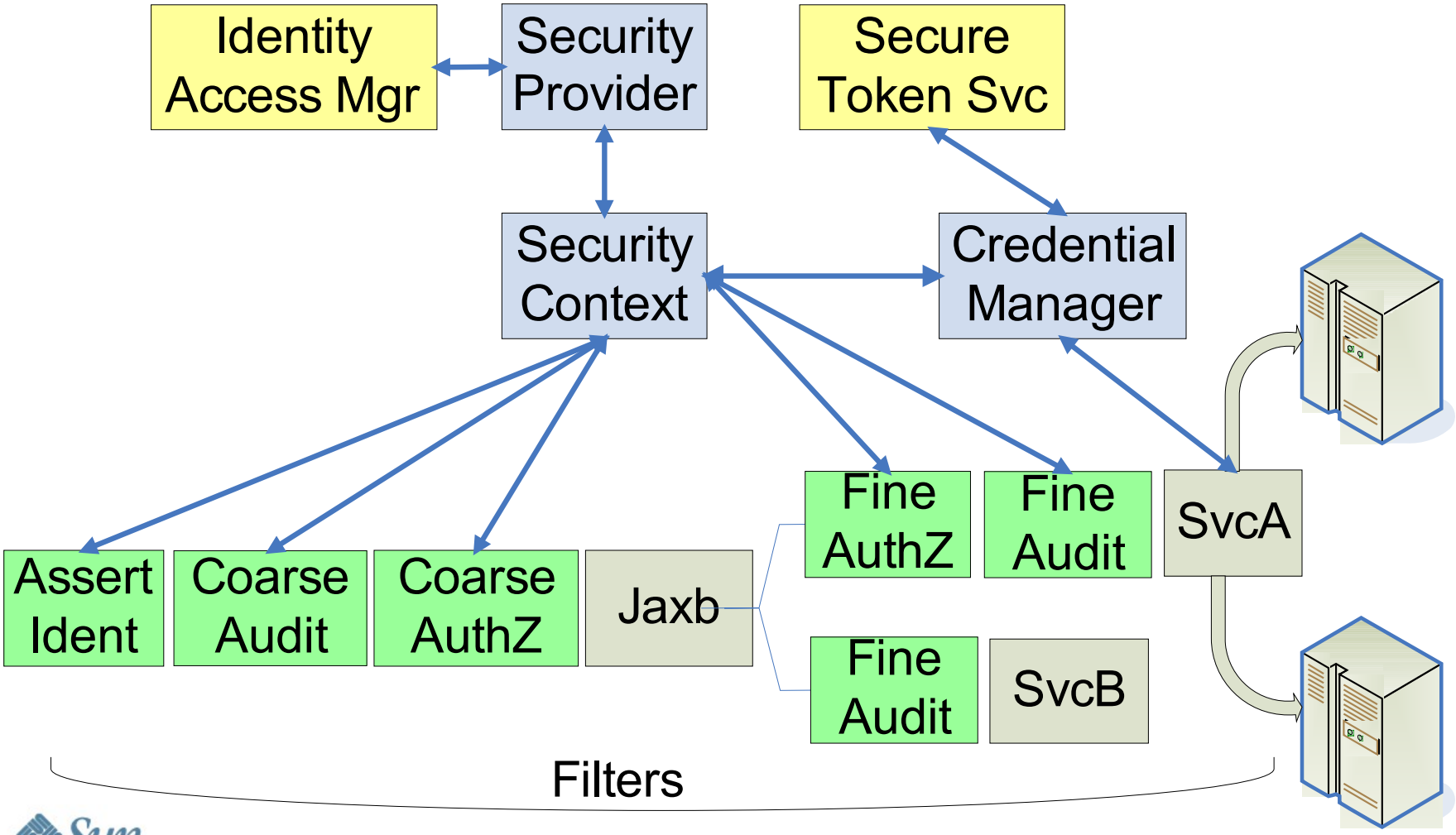
Message Architecture Overview

- Document based message interaction
  - Coarse-grained, nested message grammar
  - Message decomposed in Filter processing

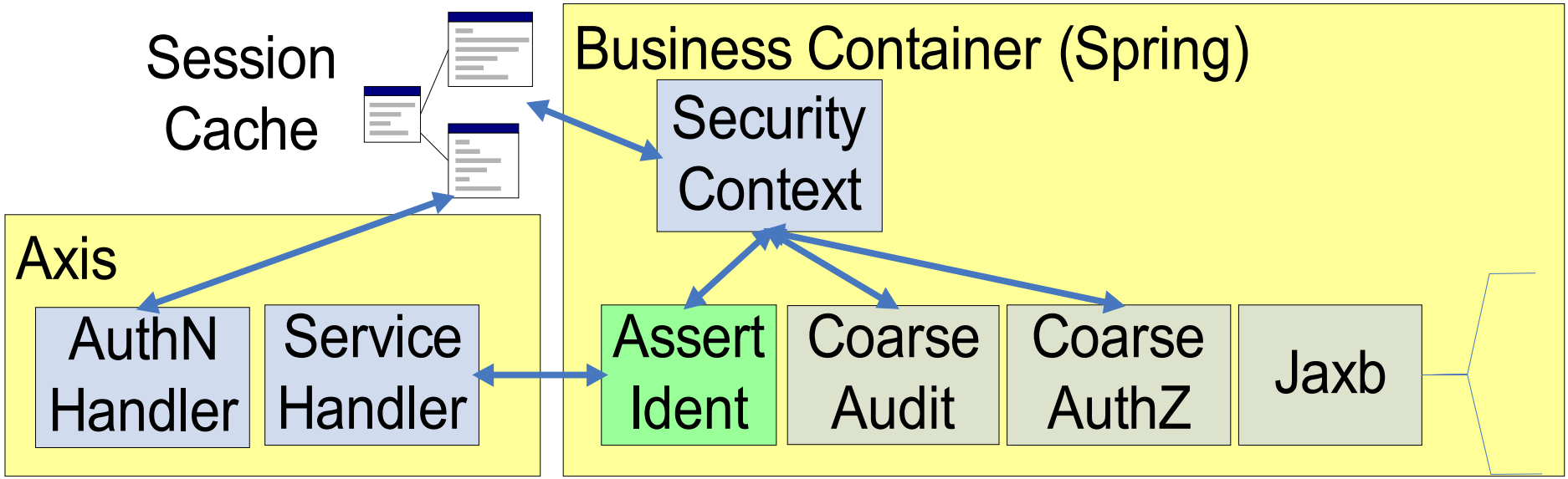
```
<IFX xmlns="http://www.ifxforum.org/ifx_150”>
   <BankSvcRq>
      <XferAddRq>
   <DepAcctIdFrom><AcctId>123</AcctId></DepAcctIdFrom>
      <LoanAcctIdTo><AcctId>456</AcctId></LoanAcctIdTo>
      <Amt>434.00</Amt>
      </XferAddRq>
   </BankSvcRq>
</IFX>
```

Business Container

Security Component Relationships



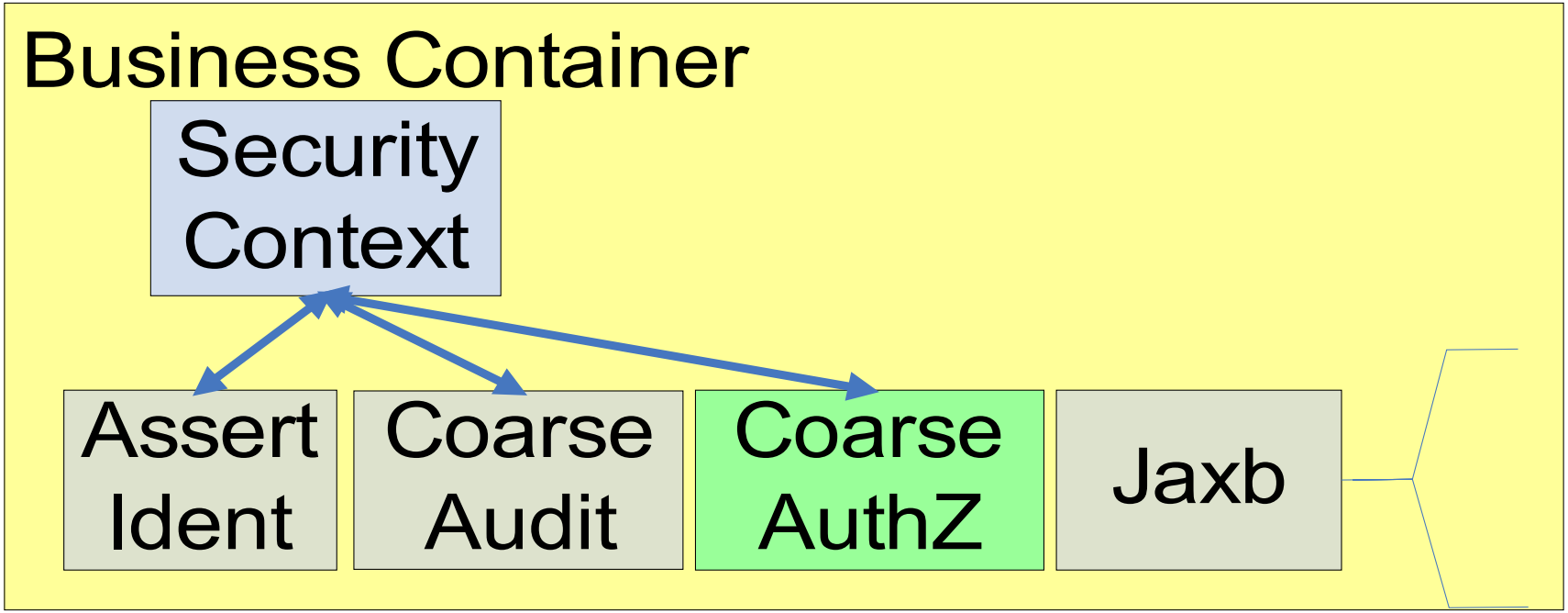
Business Container
Identity Assertion Module



- Handles identity abstraction
- Establishes Security Context

Business Container

Authorization – Coarse Grain

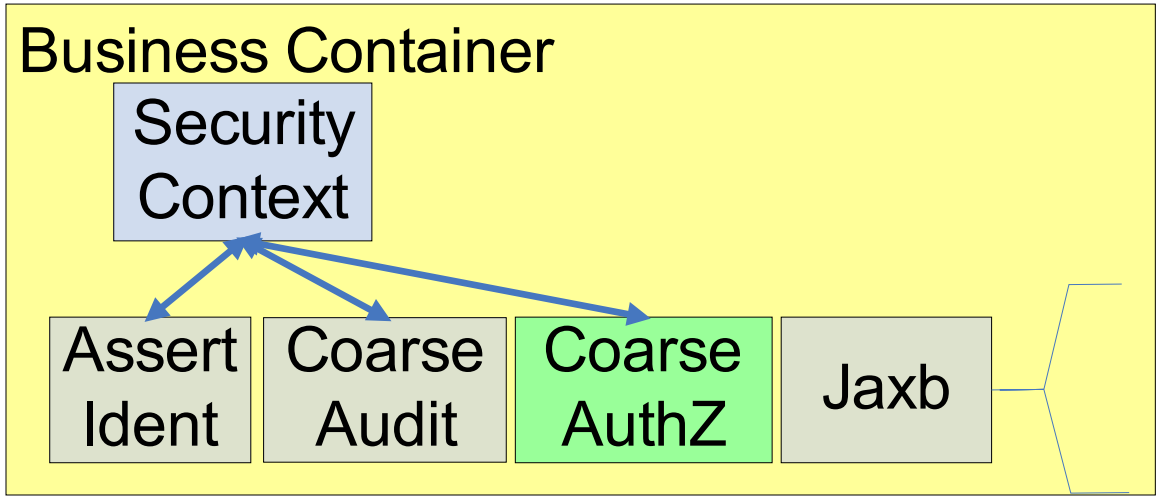


- Principle of least privilege
- Every invocation goes through this filter (fail fast)
- Little message/business knowledge

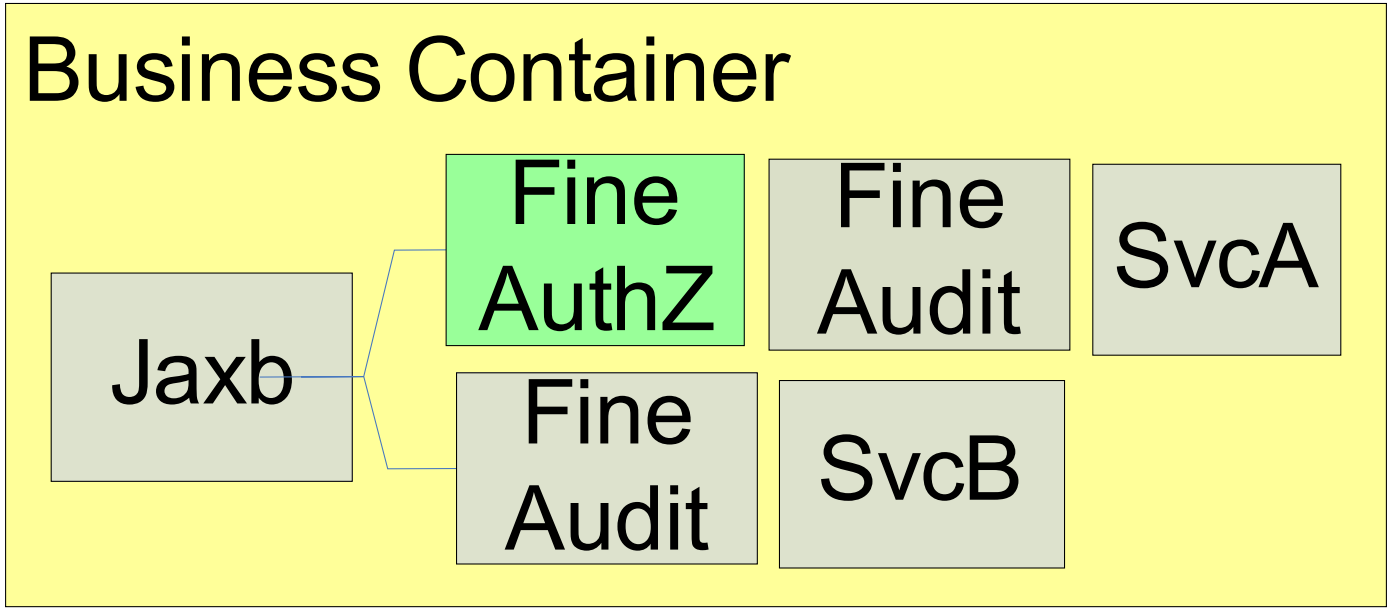
Business Container

Authorization – Coarse Grain

```
<IFX xmlns="http://www.ifxforum.org/ifx_150”>
   <BankSvcRq>
      <XferAddRq>
   <DepAcctIdFrom><AcctId>123</AcctId></DepAcctIdFrom>
      <LoanAcctIdTo><AcctId>456</AcctId></LoanAcctIdTo>
      <Amt>434.00</Amt>
      </XferAddRq>
   </BankSvcRq>
</IFX>
```



Business Container
Authorization – Fine Grain



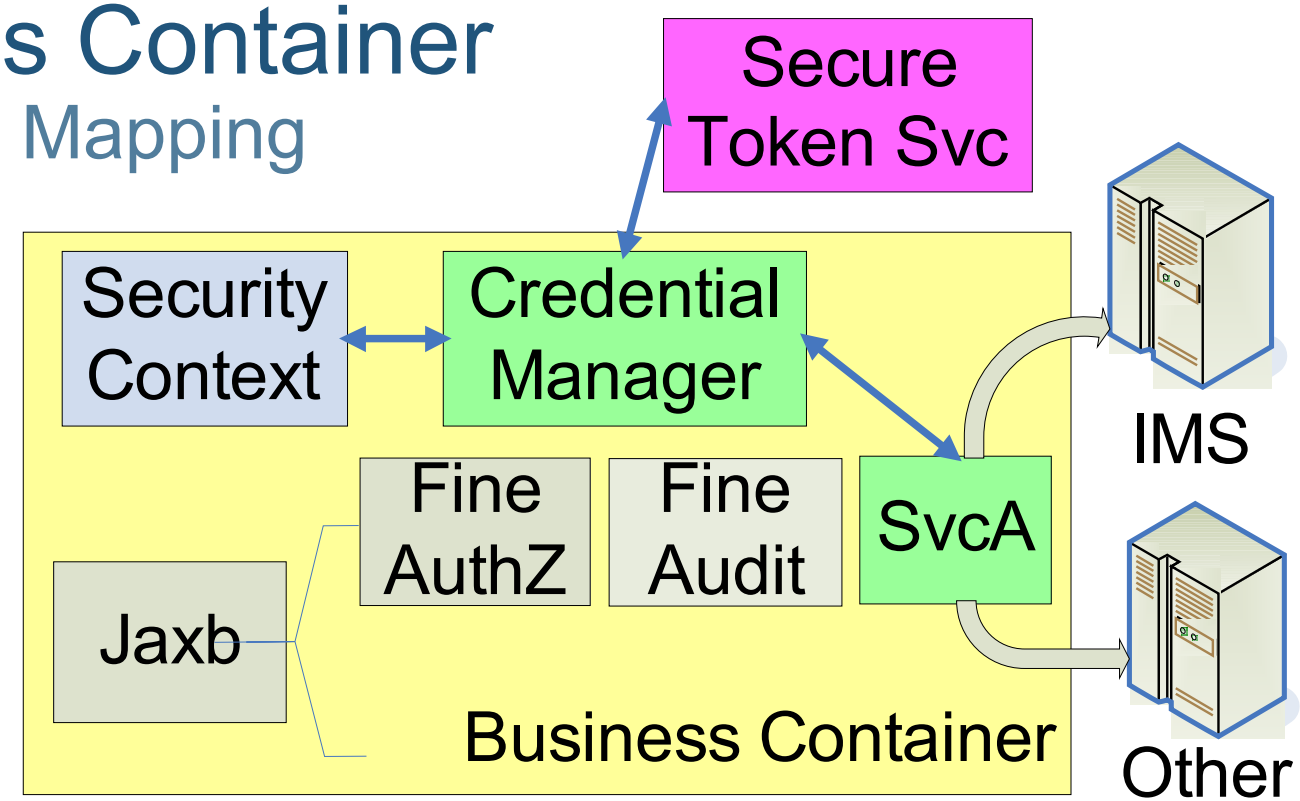
- Message specific
- Involves application data
- Costs the most so we do it last

Business Container

Authorization – Fine Grain

```
<IFX xmlns="http://www.ifxforum.org/ifx_150”>
<BaseSvcRq>
    <PermissionAddRq>
        <PermissionName>StopCKOther</PermissionName>
        <PermissionInfo>
                <Desc>Stop Check Delete</Desc>
                <OrgId>
                        <OrgIdType>OrgUnitTree</OrgIdType>
                        <OrgIdNum>CompanyA</OrgIdNum>
                </OrgId>
        </PermissionInfo>
    </PermissionAddRq>
...
```

Business Container
Credential Mapping



- Integrate with external applications
- Strategies depend on endpoint
- Can use WS-Trust enabled STS or not
- Use builder for each token type

Business Container

Credential Mapping Configuration

```
<beans>
<xyx:credbuilders class="com..trust.CredentialBuilder">

<xyy:credMap credentialClass="com...credential.SavingsCredential"
builderClass="com...credential.SavingsCredentialBuilder"/>

<xyy:credMap credentialClass="com...credential.LoanCredential"
builderClass="com...credential.LoanCredentialBuilder"/>

<xyy:credMap credentialClass="com...credential.CreditChkCredential"
builderClass="com...credential.CreditCheckCredentialBuilder"/>

</xyx:credbuilders>
</beans>
```

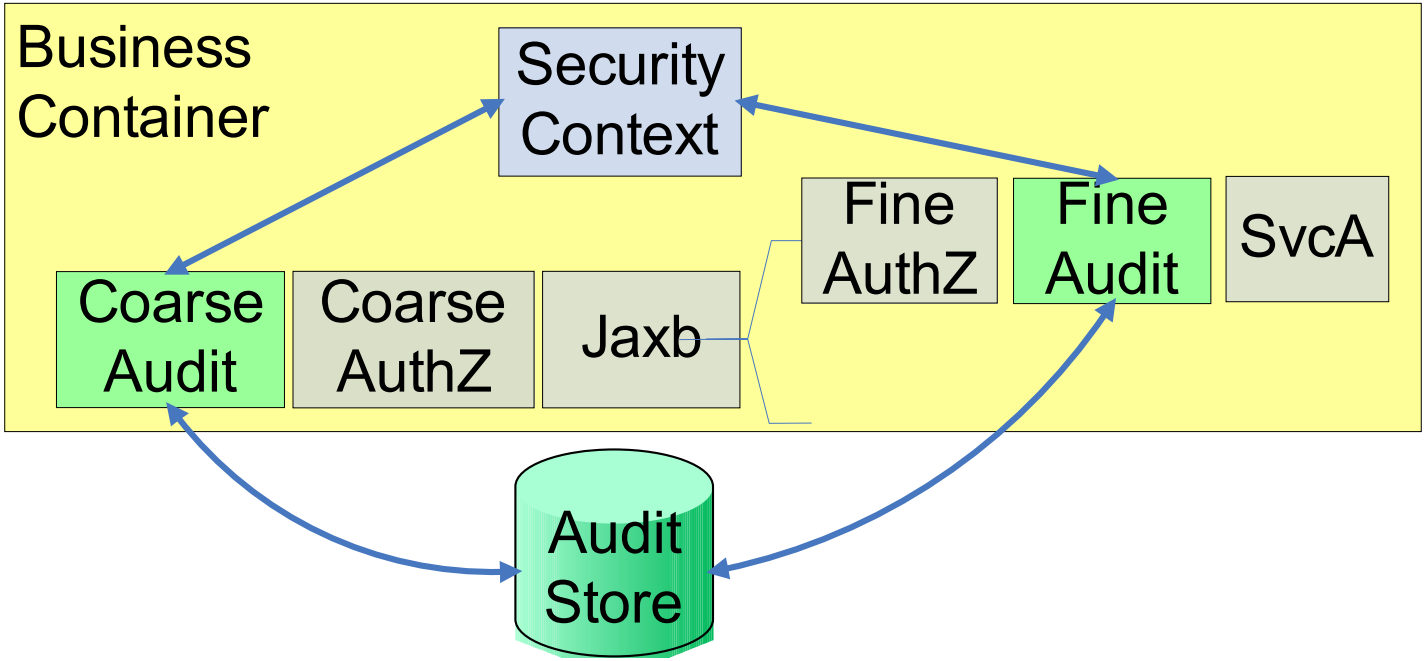
Business Container

Credential Mapping Example

```
public class XferAddSvc {
   public SecurityContext securityContext;
   public XferAddSvc (SecurityContext securityContext, ...) {
    this.securityContext =  securityContext;
   }

   public void doService(...) {
       LoanCredential loanCred =
       securityContext.getCredential(LoanCredential.class);

   }
```

Business Container

Audit Filters



- Compliance with regulations
- Need to know who did what and when
- No standard with traction – do what meets requirements

Business Container

Audit Requirements

- Audit Event Logging systems must support 3 key features:
  - An application programmer's interface or API, entirely separate from the logging facility.
  - A common, standardized audit event record format.
  - Common, standardized audit event taxonomy (a set of events with associated well-defined and documented semantic meanings).

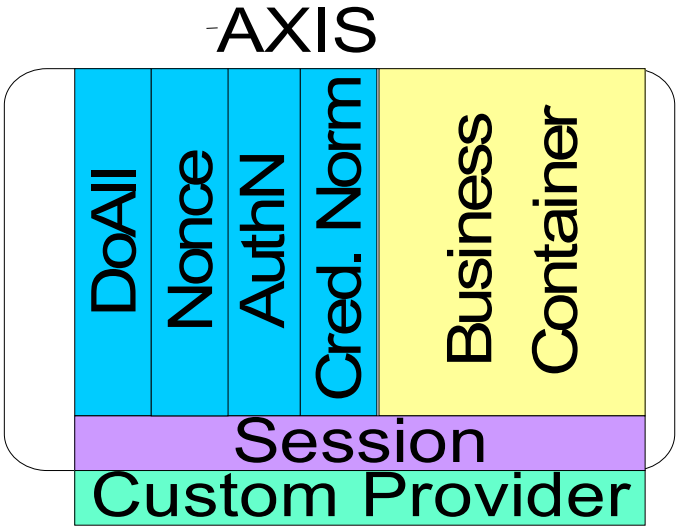
Business Container

Comparison of Standard Audit Mechanisms

Type	Industry Standard	Standard Across Vendors	API To Use	Log Syntax	Event Taxon-omy	Log Transport Available	Level Of Complex-ity	Level Of Trac-tion	Free Impl Avail-able
CEF	No	Yes	No	Yes	No	No	Low	Low	N
CEE	Pending	No	?	Yes	Yes	No	High	Med	N
xDAS	Yes	No	Yes	Yes	Yes	Yes	High	Low	Y
CBE	Yes	Yes	Yes	Yes	Yes	Yes	Med	Med	Y

CEF - Common Event Format by ArcSight
CEE - Common Event Expression language, by Mitre
XDAS - X/Open Distributed Audit Standard (XDAS), by Open Group
CBE - Common Base Event, by IBM

Security Provider



rovides authentication,
authorization,

dministrative and audit
capabilities

The reaches opened before us and closed behind, as if the forest had stepped leisurely across the water to bar the way for our return. We penetrated deeper and deeper into the heart of darkness. It was very quiet there."

oseph Conrad, *Heart of Darkness*

Security Provider
10 steps to success

1. Choose an identity repository/Password Policies
2. Implement API for Identity and Access Control
3. Implement Administration API
4. Implement SPI for J2EE Security Realm
5. Implement Provisioning Services
6. Utilize a bulk load tool for seeding IAM repository
7. Build a Provisioning GUI
8. Define your delegated admin strategy
9. Define your audit strategy
10. Maintain IAM as separate product

Security Provider

Step 1 - Choose Identity Repository

Requirements

1. Simple
2. Scalable
3. Low Cost
4. Platform Independence
5. Reliable
6. Highly Available
7. Extensible
8. Standard
9. Commercially Supported

Security Provider

Step 2 – Implement Identity and Access Control API

- Use RBAC System model for session mgmt
  - http://csrc.nist.gov/groups/SNS/rbac/documents/draf
  - 6.1.2 Supporting System Functions for Core RBAC
  - Form the basis for authentication and session mgmt

```
CreateSession(user, session)

SessionRoles(session, result)

AddActiveRole(user, session, role)

DeleteActiveRole(user, session, role)
```

Security Provider
Identity and Access Control API

- Use RBAC System model for access control
  - http://csrc.nist.gov/groups/SNS/rbac/documents/draf
  - 6.1.2 Supporting System Functions for Core RBAC
  - Form the basis for access control

```
CheckAccess(session, operation, object,
result)

SessionRoles(session, result)

SessionPermissions(session, result)
```

Security Provider
Identity and Access Control API

- Fine-grained Authorization – on your own
  - Custom API
  - Usually involves instance data which has business object smell.
  - Difficult to separate fine-grained authorization from business logic.

```
public Object getUserPermission(Subject sess,
   String permId, String attr)throws…;

public List getUserAccountPermissions(Subject
session, String accountId) throws …;
```

Security Provider
Step 3 – Administration API

- Use RBAC Administrative model
  - http://csrc.nist.gov/groups/SNS/rbac/documents/draf
  - 6.1.1 Supporting System Functions for Core RBAC
  - Form the basis for Provisioning system

```
AddUser(user)

AddRole(role)

AssignUser(user, role)

GrantPermission(object, operation, role)
```

Security Provider
Administration API

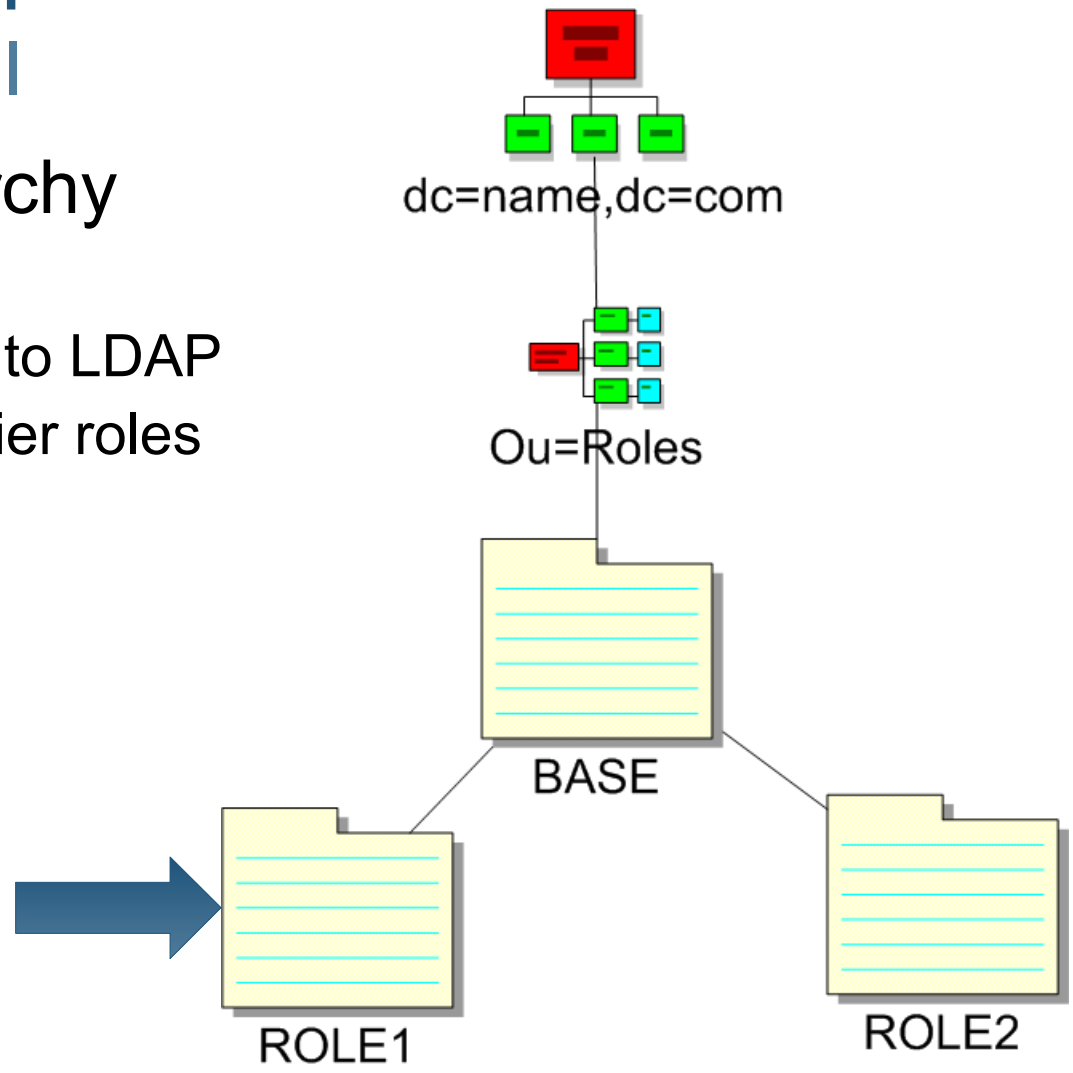
- Use RBAC Review model
  - http://csrc.nist.gov/groups/SNS/rbac/documents/draf
  - 6.1.3 & 6.1.4 Review Functions
  - Form the basis for reporting

```
AssignedUsers(role, result)

AssignedRoles(user, result)

RolePermissions(role, result)

UserPermissions(user, result)
```

Security Provider
Implement Admin API

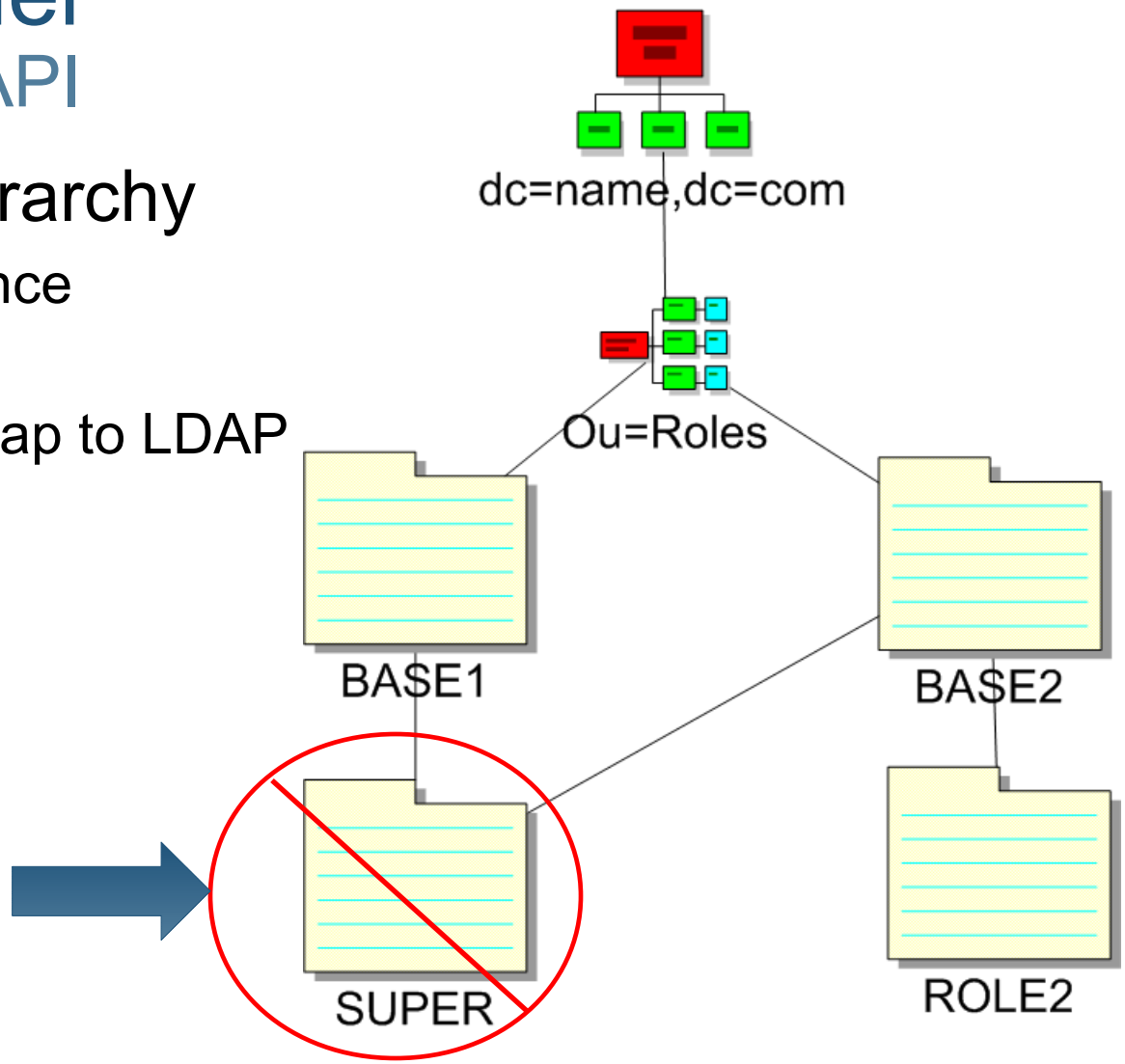
- Limited Role Hierarchy
  - Single Inheritance
  - Tree maps directly to LDAP
  - Nested objects = hier roles

Nested LDAP objects map to limited hierarchical roles



Security Provider
Implement Admin API

- General Role Hierarchy
  - Multiple Inheritance
  - More flexible
  - Graph doesn't map to LDAP



Can't Do This with LDAP

Security Provider

Step 4 - Implement SPI for Custom J2EE Security Realm

- Options
  1. Proprietary SPIs
     - Websphere – UserRegistry
     - Tomcat/Jboss – UserDatabase
     - Weblogic – Custom Security Realms
  2. Standard SPIs
     - JSR 115: JavaTM Authorization Contract for Containers
     - JSR 196: JavaTM Authentication Service Provider Interface for Containers

Security Provider
Implementing Custom Security Realm

- Pros for using Container SPI for Realm
  - Takes advantage of container specific features
  - Credential caching.
  - 80/20 rule
- Cons
  - Coarse-grained only
  - May have to maintain your own resource connection to identity repository
  - No data connection between custom realm and applications complicates password policy scenarios

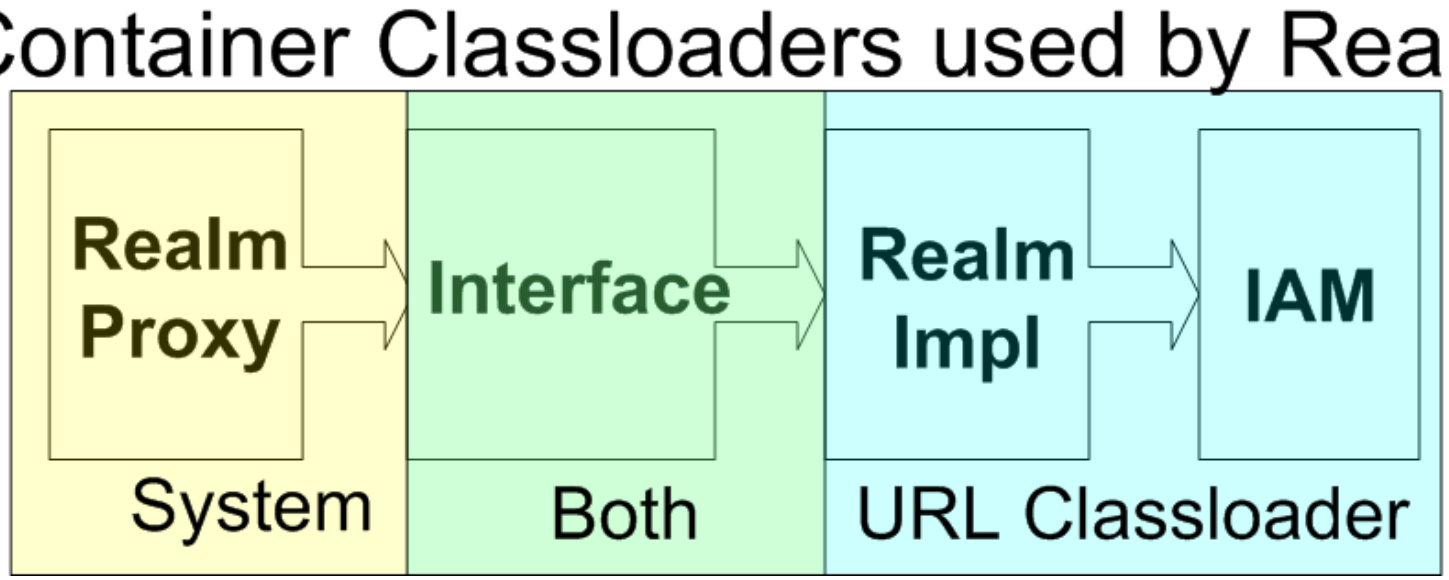
Security Provider
Implementing Custom Security Realm

- Websphere SPI:

```
public class FwUserRegistryProxy implements
UserRegistry
{
  public String checkPassword(     String
uId,
      String pswd) throws…

  public List getGroupsForUser(String uid)
           throws …
…
}
```

Security Provider

Implementing Custom Security Realm



- Realm Trick 1:
  - Isolate classes from container by accessing IAM via URLClassLoader

Security Provider

Implementing Custom Security Realm

- Realm Trick # 1

```
public class FwUserRegistryProxy implements
UserRegistry{
   private UserRegistry realm = null;
   public void initialize(Properties props){
          URLClassLoader ucl = new
             URLClassLoader(getURLs(…),
             this.getClass().getClassLoader());
          Class realmImpClass =
             ucl.loadClass(UserRealmClassname);
          realm =
           (UserRegistry)realmImpClass.newInstance();
     }
   public Result getGroups(…){
      return realm.getGroups(…);
     }
```

Security Provider
Implementing Custom Security Realm

- Realm Trick 2:
  - Utilize a pool for your database or LDAP connections.
  - Container may not make managed resources available to the realm.
  - Use pooling ability in 3rd party LDAP toolkit
    - Mozilla Java LDAP API
    - Or – use Commons Pool

Security Provider
Step 5 – Provisioning Services

- Why
  - Integration with external provisioning systems
  - Isolation of provisioning system from IAM system
- How
  - Place adapters between your service and IAM API
  - Define a Web service grammar
- Or – Utilize Standard
  - Service Provisioning Markup Language (SPML)
    - http://www.oasis-open.org/committees/download.php/17708/pstc-spml-2.0-os.zip

Security Provider
Step 5 – Provisioning Services

- Only need to implement a handful of services
  - User
    - Add, Update, Delete
  - User-Role Assignment
  - User password maintenance
- Use bulk load facility for Role-Permission maintenance

Security Provider
Provisioning Services

- Example

```
<IFX xmlns:com.fnf="http://www.fnf.com/xes"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.ifxforum.org/IFX_150">
 <com.fnf:UserAddRq>
 <RqUID>12345654-2341-9876-9283-000000000000</RqUID>
 <com.fnf:UserId>T18T18</com.fnf:UserId>
 <com.fnf:UserInfo>
 <com.fnf:PswdInfo>
 <Pswd>ABC123</Pswd>
 <com.fnf:ChannelName>TP</com.fnf:ChannelName>
 <com.fnf:PswdType>P25</com.fnf:PswdType>
 </com.fnf:PswdInfo>
 <CustId>
 <SPName>com.fnf.xes.jefs</SPName>
 <CustLoginId>T18T18</CustLoginId>
 </CustId>
 <com.fnf:UserType>RETAIL</com.fnf:UserType>
```

Security Provider
Step 6 – Utilize Bulk Load Facility

- Why
  - Necessary for seeding identity repository
  - All you need to load roles and permissions
  - Conversions
- How
  - SQL scripts or LDAP .ldif
- Or
  - Implement custom batch program using Ant
    - What we did

Security Provider

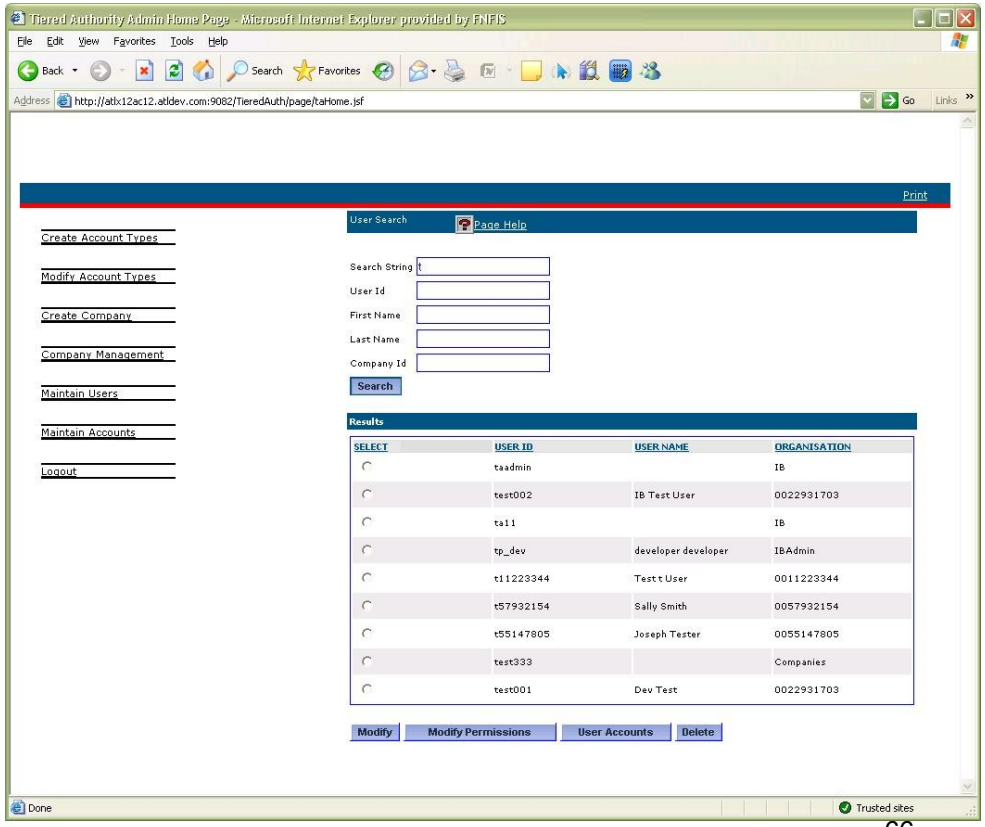
Bulk Load Facility

- Bulk Load Utility Custom Ant Task drives admin API

```
<addrole testname="Add Roles">
  <role roleNm="App1Admin" description="App Admin"/>
  <role roleNm="App1User" description="UsrRole”
</addrole>
<addpermission testname="Add Perms">
  <permission permNm=“AcctXfer” description=“”
    orgUnitId=“Dept123"/>
 <permission permNm=“AddUsr” description=“Add User“
    orgUnitId=“Dept123"/>
</addpermission>
<grantpermission testname=“Grants perms">
  <grant roleNm=“App1User" permNm=" AcctXfer"/>
  <grant roleNm=“App1Admin" permNm=" AddUsr"/>
</grantpermission>
```

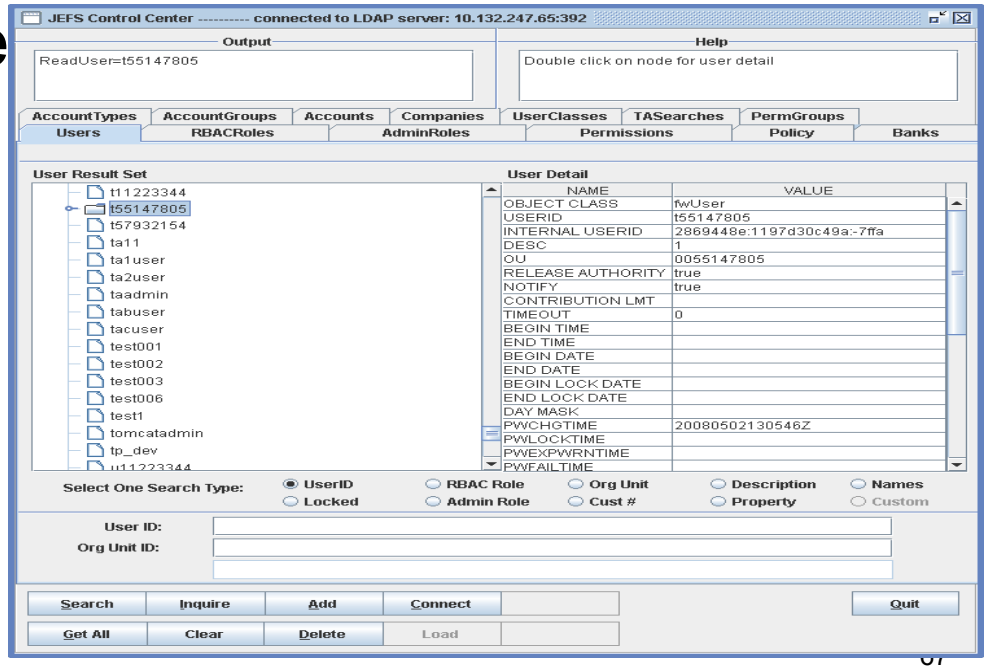
Security Provider
Step 7 - Build a Provisioning GUI

- Simple Web admin for driving admin services (3-tier)
  - User CRUD, User-Role Maintenance
  - Necessary for prod



Security Provider
Build a Provisioning GUI

- Simple Java Swing Application for CRUD
  - Useful for allowing developers and operations staff to perform routine provisioning tasks
  - Drives the Admin API (2-tier)
  - Java Web-start enable
  - Used in test envs



Security Provider

Step 8 – Utilize pswd policy extensions of your IAM repository

- Password policies are not covered by LDAP V3
- OpenLDAP implements pswd policy draft
  - http://tools.ietf.org/draft/draft-behera-ldap-password-policy/draft-behera-ldap-password-policy-09.txt
  - Password Guessing Limit
  - Password Expiration, Expiration Warning, and Grace Authentications
  - Password History
  - Password Minimum Age
  - Password Change after Reset

Security Provider
Step 9 – Define your delegated admin strategy

- Administrative Role Based Access Control
  - Use ARBAC02 Model
    - http://www.iis.sinica.edu.tw/page/jise/2006/200611_10.pdf
  - Offload administration to external departments to save costs
  - Constraints placed on what security objects administrator can access by organization.

Security Provider
Define your delegated admin strategy

- Constraints applied via filtering in provisioning services

```
public boolean canAssignRole(Subject
session, String user, String role)

public boolean canAssignPermission(Subject
session, String role, String permission)

public boolean canRevokePermission(Subject
session, String role, String permission)

public boolean isOwnedUser(Subject session,
String userId)
```

Security Provider
Step 10 – Define your audit strategy

- Options
  - Good - Instrument your Admin API with Log4j
    - Simple but reporting will suffer
  - Better - Utilize audit filter in service
    - Log request/response to database
    - Better than logger but still difficult to query data
  - Best - Store historical events via identity repository
    - Use OpenLDAP slapo access log overlay
    - logs historical events to separate DB
    - Reporting engine can pull direct via LDAP or can move into relational DB

Security Provider Review
10 steps to success

1. Choose an identity repository
2. Implement API for Identity and Access Control
3. Implement Administration API
4. Implement SPI for J2EE Security Realm
5. Implement Provisioning Services
6. Utilize a bulk load tool for seeding IAM repository
7. Build a Provisioning GUI
8. Utilize the pswd policy extensions of your IAM repository
9. Define your delegated admin strategy
10. Define your audit strategy

Security Provider
Lessons learned

- Watch out for LDAP injection attacks:
  - http://www.owasp.org/index.php/LDAP_injection
- Pay close attention to how you pool your LDAP connections.
- Maintain your security provider framework as horizontal application
- Don’t tightly bind your security data with business data.
- Learn the security tricks specific to your J2EE container.
- Ensure LDAP V3 compliance to extent possible.

Security Provider
Lessons learned (continued)

- Register for private IANA enterprise number
  - 16830 Fidelity Information Services Shawn McKinney
  - http://www.iana.org/assignments/enterprise-numbers
- Create Junit tests that exercise all of the features of your security provider.
  - Run tests automatically on every build.
  - Developers move on and you will forget all of the features added and how they work.
  - Ensures quality across release cycles.
  - QA won't know how to test.

OpenLDAP

OpenLDAP
Overview

- Efficient (FAST!)
- Scalable
- Reliable
- Stable
- Highly-available
- Portable

OpenLDAP
Performance and Scalability

- Performs well on low-cost commodity hardware.
  - Tens of thousands of Authentications and Writes a second
  - Hundreds of thousands of Searches a second
- Maximum number of objects
  - Tested to over five billion
  - No 32-bit limits
- OpenLDAP Driver for MySQL Cluster technology
  - Horizontal Scaling
  - Enhanced flexibility

OpenLDAP
High-Availability

- *slurpd* is deprecated (no longer supported)
- OpenLDAP uses “sync replication”
  - http://www.openldap.org/doc/admin24/replication.html
  - Delta sync replication available
- OpenLDAP supports Multi-Master
  - Mirror-mode (hot-failover)
  - *n-way* Multi-Master
- OpenLDAP Driver for MySQL Cluster
  - Database-managed replication and data integrity
  - Advanced reliability and performance options/capability

OpenLDAP
Architectural Flexibility: Overlays

- Used to extend capability of OpenLDAP
  - Constraints
  - Dynamic Groups
  - Proxy Caching
  - Referential Integrity
  - Uniqueness
  - Sorting
  - More - http://www.symas.com/blog/?page_id=76

OpenLDAP
Password Policies

- PPolicy Overlay
  - http://www.openldap.org/faq/data/cache/1190.html
    - Password Hashing
      - SSHA, SMD5, NTLM
    - Min/Max Age
    - History
    - Min Length
    - Expire Warning
    - Grace
    - Lockout/Reset
    - More

OpenLDAP
Database History

- Used to record all accesses to a particular OpenLDAP backend database to another database.
  - Useful to satisfy IT security audit requirements.
  - Can log authentication/authorization attempts.
  - Can be queried using LDAP V3 protocol

Summary

- Security solutions don't have to be expensive or complex.
  - Isolate your applications from your security providers.
  - Use of standard and open solutions can reduce cost of deploying Web applications by removing dependency on commercial AAA vendors.

More Information

- Download blueprint:
  - http://www.webaccessmanager.com/wiki/

JavaOne℠

Thank You

Shawn McKinney
shawn.mckinney@fnis.com
Mike Scheuter
mike.scheuter@fnis.com
Marty Heyman
mheyman@symas.com