SECURITY

Avoid the danger which has not yet come

EISA – the BIG picture

- **Enterprise information security architecture (EISA)**

  the part of the **enterprise architecture** focusing on information security throughout the enterprise.

  Key component of the information security technology governance process at any organization of significant size

The Bottom Line

Fundamentally:

Most of the damage to Information Technology (IT) security is not from outside malicious attacks, but rather from simple mistakes, unintended **or unauthorized actions** of legitimate users and IT engineers who are either untrained in security and/or who misunderstood the instructions from the management.

Benson Yeung, Senior Partner – TNS

…biggest problem is simple stupidity. **Lax access policies** and internal employees are still the biggest source of security headaches. A recent Verizon report backs this up, with research indicating that 96 percent of attacks are not sophisticated and 97 percent are easily avoidable.

eWeek - Top 10 Security Issues

Fundamentally:

Keep It Simple…

applies well to security - Complexity increases the risk of problems. Avoid complexity and avoid problems.

The Spring Fundamentals

Authentication

refers to unique identifying information from each system user, generally in the form of a username and password.

Authorization

refers to the process of allowing or denying individual user access to resources.

Spring provides for externalized authorization

Access Control List

every secure domain object has an associated definition[ACL] that defines domain object specific authorization [CRUD]

These concepts are industry wide , and not at all specific to Spring

NOTE: Core Enterprise Security is basically communication technology agnostic [JMS,RMI, Flat File, etc.]

[RE: it is NOT HTTP specific]

Spring Security

- Provides authentication and authorization for enterprise applications.
- These are the two main areas that Spring Security targets.
- Base on 'Acegi Security' (pronounced *Ah-see-gee*)

Brief History

Original Acegi 2003

Apache.org 2004

Spring Project 2006

Spring Security 2008

Spring Security Core Classes

- **SecurityContextHolder** helper class to provide access to the SecurityContext.
- **SecurityContext** holds the Authentication Object [current authenticated user].
- **Authentication** represents the principal in a Spring Security-specific manner.
- **GrantedAuthority** reflects the application-wide permissions granted to a principal.

- **Helper Interfaces:**
- **UserDetails Interface** provides the necessary information to build an Authentication object from your application's “custom” DAOs or other source of security data.
- **UserDetailsService Interface** loads & creates a “custom” UserDetails when passed in a String-based username.

Main Point

- Security underlies an entire enterprise. It provides a shield that makes the application invulnerable.

- ***Science of Consciousness:*** *Transcendental Consciousness is characterized by the quality of invincibility, which means one cannot be overcome or overpowered.*

Authentication

- Spring supports a wide range of authentication models:

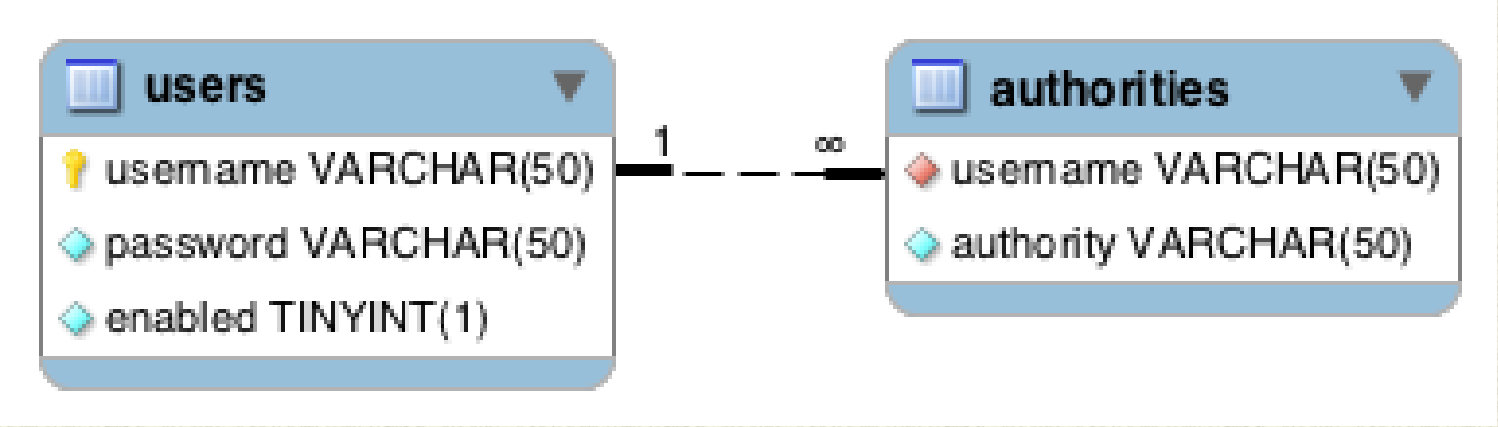
- HTTP BASIC
- HTTP Digest
- HTTP X.509
- Java Open Source Single Sign On (JOSSO)
- Form-based
- Java Authentication and Authorization (JAAS)
- Central Authentication Service (CAS)
- Automatic "remember-me
- Anonymous authentication
- Run-as authentication

Atlassian Crowd
Roller
Tapestry
LDAP[Lightweight Directory Access Protocol]
Kerberos
OpenID
Mule ESB
Elastic Path
AppFuse
Oauth2

Spring Standard Authentication Tables

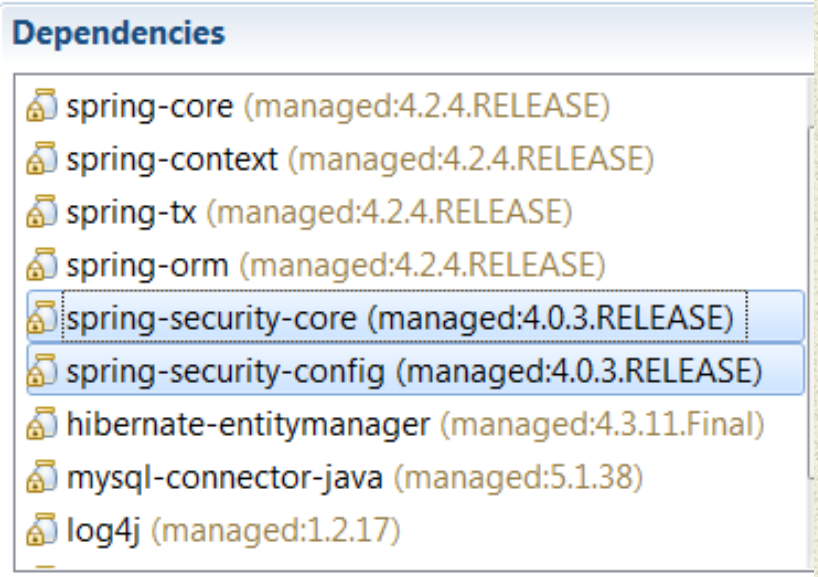
- These standard JDBC implementation tables to load the password, account status (enabled or disabled) and a list of authorities (roles) for the user.
- Each user can have zero or more “authorities”, or roles

Standard JDBC implementation of the UserDetailsService (JdbcDaoImpl)



- Authority as Roles: ROLE_USER,ROLE_ADMIN…
- Authority [Authorization] to do “something”

- **Maven Dependencies**
-



- **XML NameSpace**
- `xmlns:security="http://www.springframework.org/schema/security"`
- *http://www.springframework.org/schema/security*
  http://www.springframework.org/schema/security/spring-security.xsd

Spring Standard Authentication Provider Configuration

```
<!--  Enable AOP Authorization Annotations -->
 <security:global-method-security pre-post-annotations="enabled"  >
</security:global-method-security>
 <!-- "custom" queries are because we have used our own schema...-->
<security:authentication-manager alias="authenticationManager">
   <security:authentication-provider>
       <security:password-encoder hash="bcrypt" />
       <security:jdbc-user-service
              data-source-ref="dataSource"
       users-by-username-query="select username,password,enabled
                              from credentials where username=?"
       authorities-by-username-query="select u1.username,
              u2.authority from credentials u1, authority u2 where
              u1.username = u2.username and u1.username =?" />
   </security:authentication-provider>
</security:authentication-manager>
```

Custom “standalone” Authenticate a user [login]

- **AuthenticateUser.java**

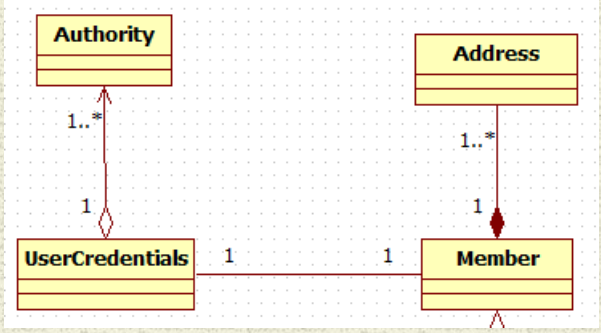
```
authenticationManager DECLARED in Config
```

```
Authentication request = new
            UsernamePasswordAuthenticationToken(name, password);
Authentication result =
                    authenticationManager.authenticate(request);
SecurityContextHolder.getContext().setAuthentication(result);
```

- **To Logout**

```
// Clears the context for the current user/thread
SecurityContextHolder.clearContext();
```

Code Example




```
Authority authority = new Authority();
    authority.setAuthority("ROLE_USER");
    authority.setUsername("Sean");

    UserCredentials userCredentials = new UserCredentials();
    userCredentials.setUsername("Sean");
    userCredentials.setPassword("Sean");
    userCredentials.setEnabled(true);

    userCredentials.getAuthorities().add(authority);
    Member member = new Member();
    member.setFirstName("Sean");
    member.setLastName("Smith");
    member.setMemberNumber(1);

     member.setUserCredentials(userCredentials);
    memberService.saveFull(member);
```

SEE SecurityAuthentication DEMO

AUTHORIZATION

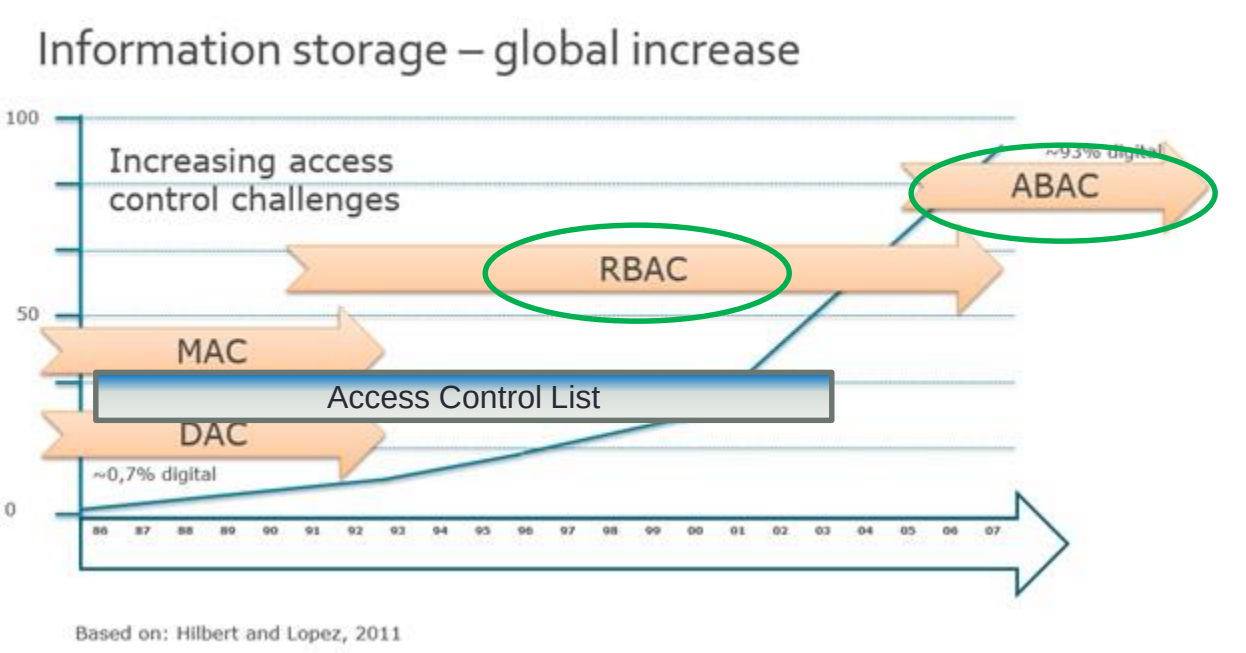
Authorization
Access Control Mechanisms

Mandatory Access – Rigid Fixed; Military

Discretionary Access – User/Admin modification of access permission
"usually" ACL based

Role Based – centered around ogranizational structure role-permissions, user-role and role-role relationships
"can" be ACL based

Attribute Based – Attributes (user, resource, object, environment) dynamically evaluated by policies [rules]



RBAC
Role Based Access Control

Common Enterprise approach to managing users' access to resources

Basic Principle:

Access is through **Roles** - **Permissions** are given to **Roles**, not to **Users**.

Users are assigned to **Roles.**

Permissions are given to a **Roles**.

User has one or more **Roles**.

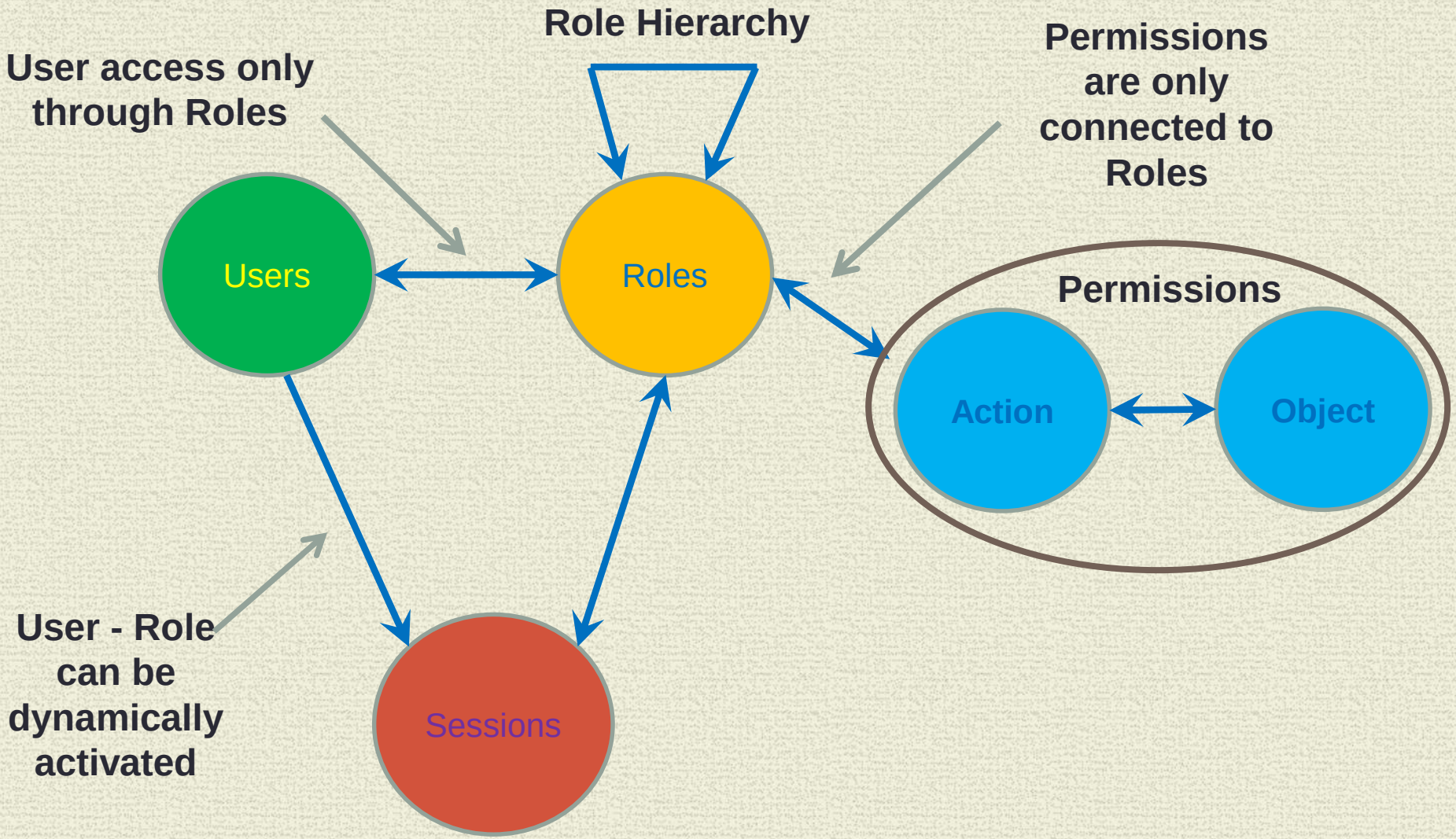
RBAC has the following model:

Many To Many relationship between **Users** and **Roles**.

Many To Many relationship between **Roles** and **Permissions**.

Roles can have a parent **Role**. {Hierarchy – advanced RBAC}

RBAC Model


Role Hierarchy
User access only through Roles
Permissions are only connected to Roles
Users
Roles
Permissions
Action
Object
User - Role can be dynamically activated
Sessions

RBAC Applications

- Addresses needs of commercial and government organizations. Used to facilitate administration of security in large organizations with hundreds of users and thousands of permissions.
- **However**
- User-centric does NOT consider attributes such as :

  resource information

  relationship between user and the resource

  dynamic information [e.g. time of the day or user IP].

- ABAC addresses this by defining access control based on attributes **RBAC is essentially a subset of ABAC with one attribute [Role].**
- **See** NIST - RBAC

Attribute Based Access Control [ABAC]

ABAC is a "next generation" authorization model that provides dynamic, context-aware and risk-intelligent access control.

"By 2020, 70% of all businesses will use ABAC as the dominant mechanism to protect critical assets, up from 5% today."

Gartner Predictions [2013]

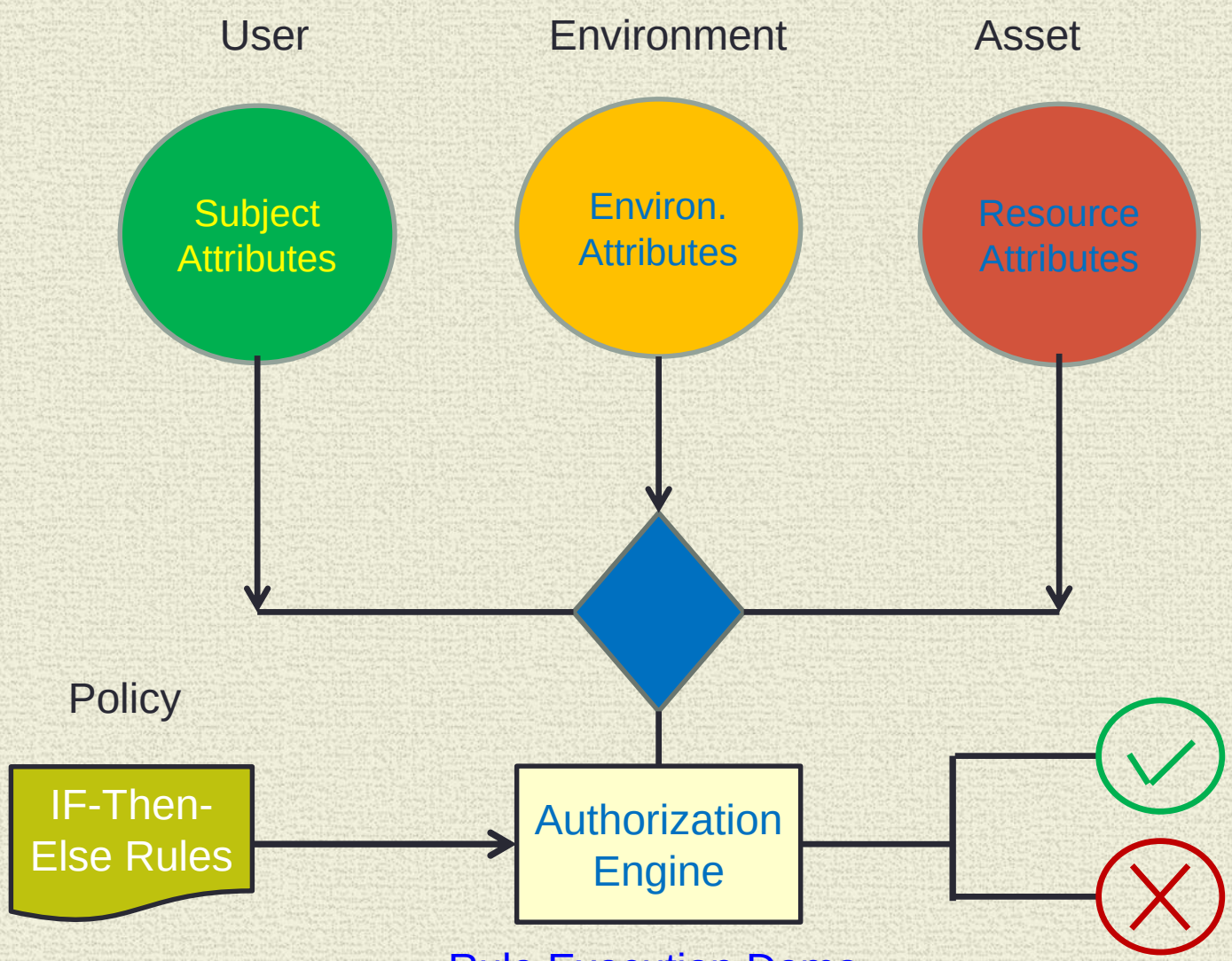
ABAC defines access by matching the current value of

user attributes, **object attributes**, and **environment conditions**

with

requirements specified in **access control policies[rules**]

ABAC "Architecture"



Rule Execution Demo

ABAC Scenario

- **ATTRIBUTES:**
- **User:** Group Memberships, Department ,Management level
- **Action:** Read, etc.
- **Asset:** Debit Account, Health Record, etc…
- **Environment:**
  Time; Physical Location; Device type [Smart phone, PC]

- **Policy [Rule]**
- All **Employees** belonging to the **Product Department** should have **full access** to **uploaded products** between **9 am and 10 pm M-F**.

ABAC Applications

- **Policy** & **Attributes** can be defined in a Technology neutral way

  Microservices, Applications,

  Database, Big Data…

  InternetofThings

- Implementation through
- XACML [eXtensible Access Control Markup Language]

  standard for attribute- and policy-based access control

- **However**
- Auditing is difficult with ABAC …
- The rules and attribute management can get complex..

Simple Spring "Role" Based * Method Level Authorization Example

Enable annotations & expression-based access control

- Configuration:

```
<security:global-method-security
                    pre-post-annotations="enabled"/>
```

Allows logic to be encapsulated in expression.

- MemberServiceImpl.java

```
 @PreAuthorize("hasRole('ROLE_USER')")
  public void findAll() {
      memberDao.findAll();
```

"SIMPLE" is keyword – not very scalable or maintainable

- * But ***NOT*** RBAC

SEE Authentication Demo [again]

Common built-in expressions

Expression	Description
hasRole([role])	Returns true if the current principal has the specified role.
hasAnyRole([role1,role2])	Returns true if the current principal has any of the supplied roles (given as a comma-separated list of strings)
hasAuthority([authority])	Returns true if the current principal has the specified authority.
hasAnyAuthority([auth,auth])	Returns true if the current principal has any of the supplied authorities (given as a comma-separated list of strings)
principal	Allows direct access to the principal object representing the current user
authentication	Allows direct access to the current Authentication object
permitAll	Always evaluates to true
denyAll	Always evaluates to false
isAnonymous()	Returns true if the current principal is an anonymous user
isRememberMe()	Returns true if the current principal is a remember-me user
isAuthenticated()	Returns true if the user is not anonymous
isFullyAuthenticated()	Returns true if the user is not an anonymous or a remember-me user
hasPermission(Object target, Object permission)	Returns true if user has access to target for the given permission
hasPermission(Object targetId, String targetType, Object permission)	Returns true if user has access to target for the given permission

Spring ROLE HIERARCHY

USE CASE:

ROLE_ADMIN should contain ROLE_USER privileges.

Solutions[s]:

Admin user :== ROLE_ADMIN, ROLE_USER

OR

hasAnyRole('ROLE_USER','ROLE_ADMIN')

OR

Configure ROLE HIERARCHY:

In *Authentication* DEMO, Bill has Admin Role . He cannot execute a Member findAll() because it requires ROLE_USER!

```
<bean id="roleHierarchy"
class="org.springframework.security.access.hierarchicalroles.RoleHierarchyImpl">
  <property name="hierarchy">
   <value>
      ROLE_SUPERVISOR > ROLE_ADMIN
      ROLE_ADMIN > ROLE_USER
      ROLE_USER > ROLE_GUEST
   </value </property>
</bean>
```

With Role Hierarchy Admin Role will Assume User Role

Bill can now execute a Member findAll()

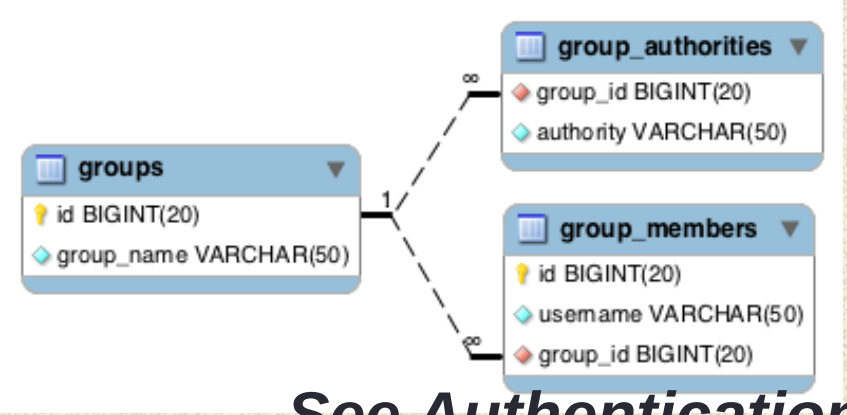
Still "SIMPLE"– not very scalable or maintainable

Hierarchy DEMO

Spring Security Groups

- Users and Roles are structured according to Groups
  - Users are organized into groups;
  - Roles are assigned to the groups.
  - User inherits Roles from Associated Group[s].
- Reflects an enterprise organizational structure
- Users can be moved in/out of a group “at will”…

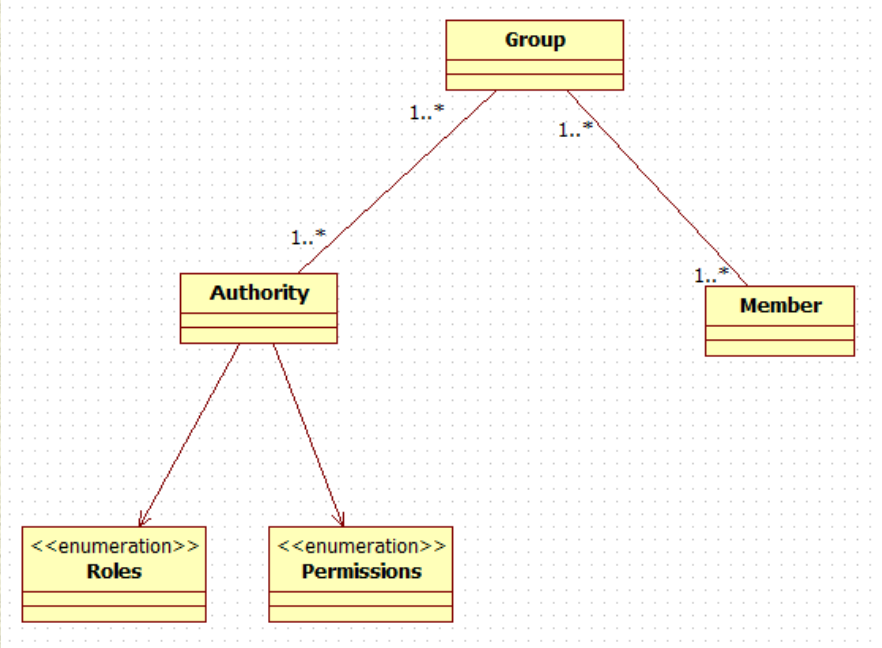
Spring Standard Authority Groups Tables



- ***See AuthenticationGroup Demo***

Basic RBAC Support in Spring

1. Many To Many Relationships
2. No ***direct*** Member to Permission Assignment
3. "Active" [Based on User session] Group Filtering **
4. Hierarchy of Groups [= RBAC Roles] **



**** Can be achieved through custom implementation**

**** Advanced RBAC - can be achieved using LDAP**

Permission Based Access

As a general rule, prefer permission-based access to role-based access. There are exceptions to this rule, but it holds as a general rule.

Maintaining correct Authorization [Adding & Deleting ROLES] is an ongoing code level task

```
@PreAuthorize("hasAnyRole('ROLE_USER',‘ROLE_ADMIN','ROLE_SUPERVISOR')")
        public Member getMember(long id){ ... }
```

Assume ROLE_s NOT hierarchical

VERSUS

```
@PreAuthorize("hasAuthority('READ')")
        public Member getMember(long id){ ... }
```

This is more maintainable since access is based on READ permission. Just remove permission from GROUP instead of modifying inline annotations

See AuthGroupPermission DEMO

Custom Rule Authorization

- Authorization can get more complicated than simple allow/deny access.
- Authorization decisions need to be made concerning the actual domain object

Fine grained object Control

The built-in hasPermission() expression is linked into the Spring Security ACL

We can create a

Custom implementation of PermissionEvaluator allows for custom rules

Override ACL permissionEvaluator

```
• <security:global-method-security pre-post-annotations="enabled"  >
•     <security:expression-handler ref="methodExpressionHandler"/>
• </security:global-method-security>
•
<bean id="methodExpressionHandler"
        class="org.spring...DefaultMethodSecurityExpressionHandler">
•     <property name="permissionEvaluator">
•       <bean id="permissionEvaluator"
•         class="edu.mum.security.rules.CustomPermissionEvaluator"/>
•     </property>
• </bean>
```

Rule Execution Demo Implementation

```
@PreAuthorize("hasPermission(#comment,'update')")
  public void update(Comment comment)
```

```
public class CustomPermissionEvaluator implements PermissionEvaluator
   @Override
   public boolean hasPermission(Authentication authentication,
                                Object targetDomainObject, Object permission) {
 // Setup "environment" values
environment.put("timeZone" , authenticateUser.getTimeZone());
// Look up asset specific policy/rules
Policy policy= (Policy)
  Main.policyList.get(targetDomainObject.getClass().getSimpleName());
// policy is ABAC context [User, action, asset, environment]
  return policy.checkRules(authentication, (String)permission,
                                   targetDomainObject, environment);
```

ABAC Architecture

Spring & ABAC

- Spring Security ABAC Support
  - Utilizes cross-cutting concern
  - Separates application logic from rule logic.
  - Has the capability to support XACML

- **By Example: ****
- Axiomatics[3rd party vendor] provides an SDK for integrating XACML into Spring Security Enterprise Java applications.

  Axiomatics -Enhancing Spring Security
- The SDK provides three core capabilities:
  - URL level access control using Web Expressions **WEB**
  - UI level access control using JSP Tag library **WEB**
  - ***Method level access control (PreAuthorize, PostAuthorize, PreFilter and PostFilter)***
- ****NOTE: Essentially an XACML implementation of hasPermission.**

Summary
Access Control on Service Methods

- **Check for Role [Spring definition of Role]** **"SIMPLE"**

```
@PreAuthorize("hasRole('ROLE_ADMIN')")
  public void delete(Comment comment) {
```

- **Check for Permission[AKA Authority]** **Use with groups [RBAC]**

```
@PreAuthorize("hasAuthority(DELETE')")
  public void delete(Comment comment) {
```

- **Check against [custom] Rule** **Fine grained [ABAC]**

```
@PreAuthorize("hasPermission(#comment,'update')")
  public void update(Comment comment)
```

SpEL syntax
#comment refers to method argument
NAME
it is checking for
`Update Permission`

Main Point

1. Spring Security allows for authentication and authorization of system users. It gives access to resources “appropriately”. It works as a stabilizing factor in the enterprise infrastructure
2. ***Science of Consciousness:*** *Securing life at its basis - at the underlying field of Creative Intelligence – guarantees stability and success at all levels*