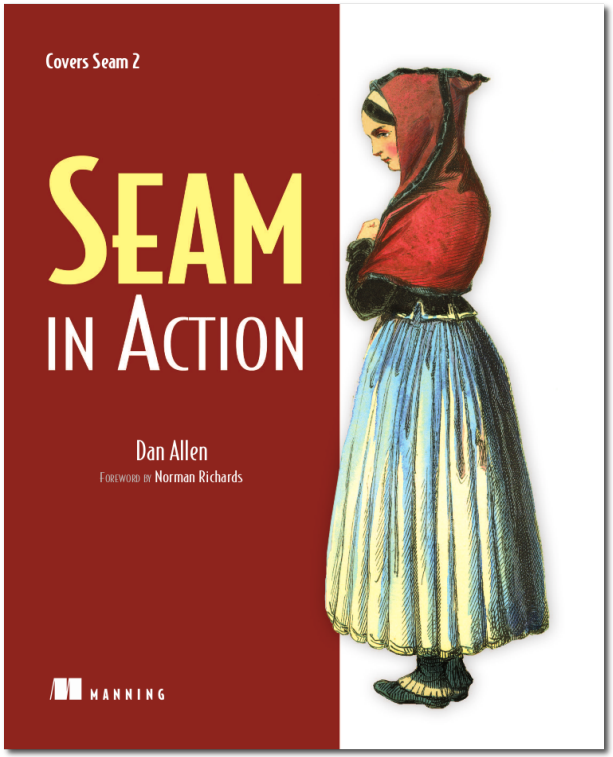
Maturing your application's security with Seam Security

Dan Allen
Senior Software Engineer
JBoss, by Red Hat

Who am I?

- Author of Seam in Action, Manning 2008
- Seam and Weld project member
- JSR-314 (JSF 2.0) EG member
- JSF user from the trenches

mojavelinux.com
. : OPEN SOURCE ADVOCACY : .



Objective

To discover the wide array of options for securing your application with Seam security

Agenda

- Security principles
- Authentication (in 3 steps)
- Identity management
- Open ID
- Authorization
- Permission management

Security needs

- It should be *easy* to setup
- It should be *easy* to manage
- The application should not outgrow it

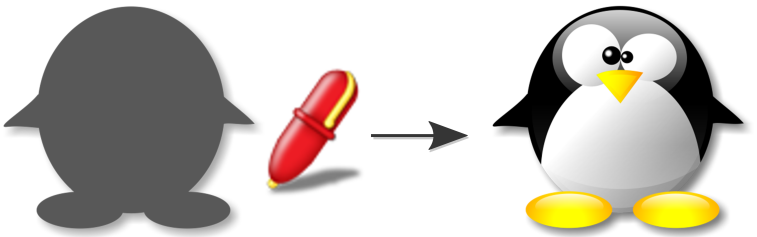
JAAS

- A surviving remnant of J2EE
- Obscure and complex configuration
- Too container dependent
- Poorly documented
- Pluggable? At what cost?

Let's get back to basics

Security principles

- **Identity**
  - Principle – who you are
  - Grants – roles and groups
- **Authentication**
  - Proving you are you
  - Relies on your secret
- **Authorization**
  - Who shall pass
  - Resource control



Authentication

Authentication with Seam in 3 steps

1) Declare an authentication method
2) Create a JSF login form
3) Write the authentication method

Step 0: No prerequisites

- Security is a core concern in Seam
- Authentication
  - Preemptive: Login API
  - Lazy: Built-in redirection to and from login page
- Authorization
- Identity and permission management
- Out of the box security setup in seam-gen projects

Demo: Security setup in seam-gen

Step 1: Declare an authentication method

- Method requirements
  - No arguments
  - Return boolean if credentials are valid
  - Must be accessible via the EL
- *No special interfaces!*
- Declared in Seam component descriptor

```
<security:identity authentication-method="#{authenticator.authenticate}"/>
```


Step 2: Create a JSF login form

- Native JSF login form
  - Bind credentials to inputs
  - Wire form action to identity component's login method
- Built-in "remember me" support

```
<h:form id="login">
    <h:panelGrid columns="2">
        <h:outputLabel for="username">Username</h:outputLabel>
        <h:inputText id="username" value="#{credentials.username}"/>
        <h:outputLabel for="password">Password</h:outputLabel>
        <h:inputSecret id="password" value="#{credentials.password}"/>
        <h:selectBooleanCheckbox id="rememberMe"
            value="#{rememberMe.enabled}"/>
    </h:panelGrid>
    <div><h:commandButton value="Login" action="#{identity.login}"/></div>
</h:form>
```


Step 3: Write the authentication method

- Basic procedure:
  - Credentials captured from login form are injected
  - Application validates credentials
  - Grant roles using identity component

```
@Name("authenticator")
public class Authenticator {
    @In Identity identity;
    @In Credentials credentials;
    @In EntityManager entityManager;
    public boolean authenticate() {
        User u = (User) entityManager.createQuery("...").getSingleResult();
        if (u.getPassword().equals(credentials.getPassword()) {
            identity.addRole("member");
            return true;
        }
        return false;
    }
}
```


Identity management

Turning authentication over to Seam

- Pluggable identity store (JPA and LDAP)
- Annotation-based identity mapping

```
public @Entity class UserAccount {
    public @UserPrincipal String getUsername() { ... }

    public @UserPassword(hash = "MD5") String getPasswordHash() { ... }

    public @UserRoles @ManyToMany Set<UserRole> getRoles() { ... }
}
```

```
public @Entity class UserRole {
    public @RoleName String getName() { ... }
}
```

- Classes designated on identity component

```
<security:jpa-identity-store
    user-class="com.company.app.model.UserAccount"
    role-class="com.company.app.model.UserRole"/>
```


API for managing identities

- UserSearch
  - Populates data model of users and handles user selection
- UserAction
  - Manages conversation for adding or modifying selected user
- RoleSearch
  - Populates data model of roles and handles role selection
- RoleAction
  - Manages conversation for adding or modifying selected role

Demo: Identity management

Open ID

Delegating authentication to a third party

- Open ID
  - Eliminates need for multiple usernames across sites
  - Users gets to choose who to trust with their credentials
  - You don’t have to maintain authentication secrets
- Seam has a built-in openid component
  - Negotiates with third party to assign user an identity principal
  - Used in place of identity component on login page
- You may still want to create a local profile for the user
  - Redirect new user to registration page after first login

Demo: Open ID

Open ID setup

- Add phase listener for handling callback from provider

```
<phase-listener>org.jboss.seam.security.openid.OpenIdPhaseListener</phase-listener>
```

- Add Open ID libraries and dependencies to classpath
- Create login page and configure navigation rules

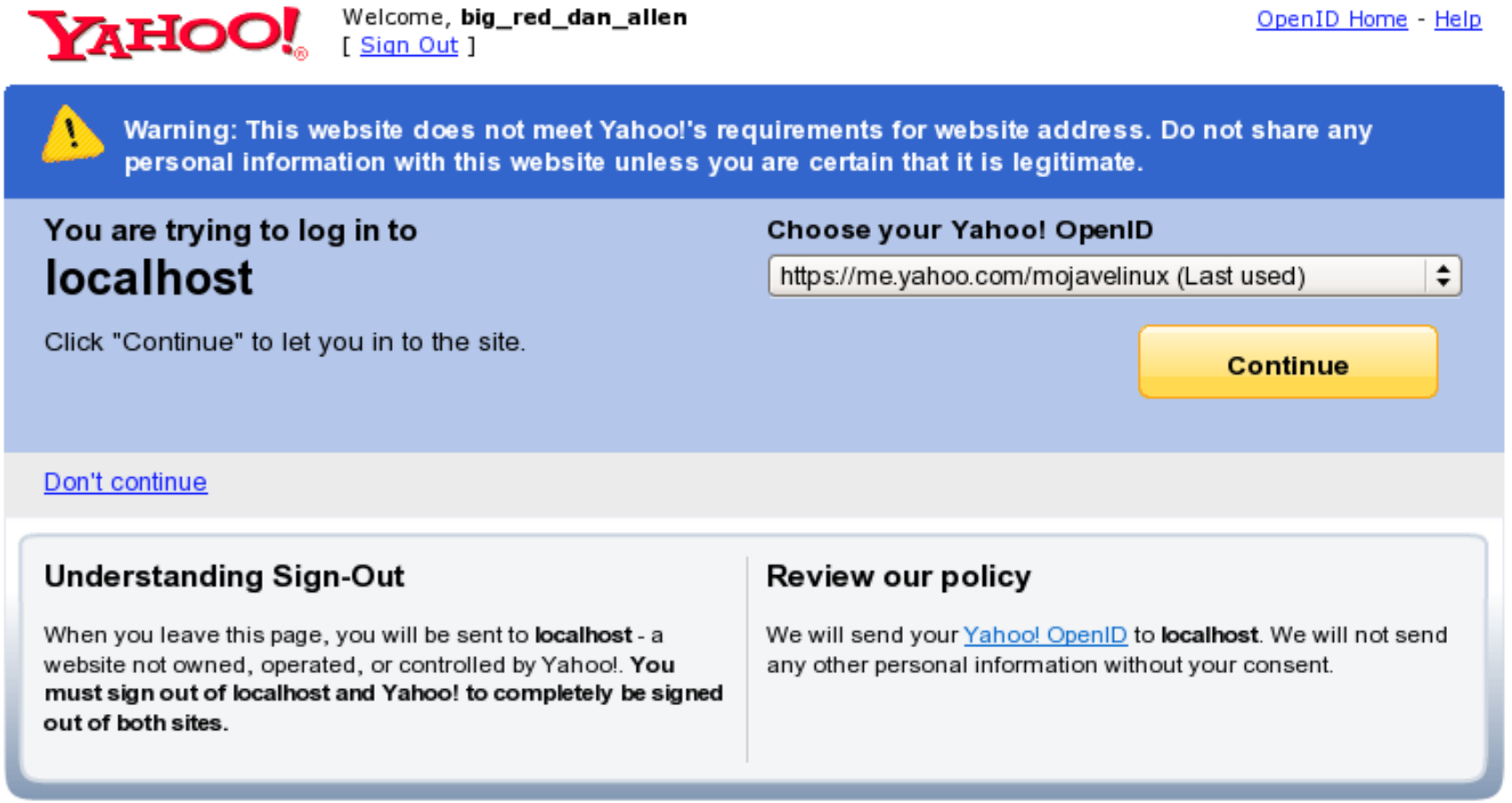
Open ID login page

- User chooses provider (AOL, Blogger, Yahoo, etc)
- Seam negotiates hand-off (using openid4java)

```
<h:form id="login">
    <h:outputLabel for="openid">Open ID</h:outputLabel>
    <h:inputText id="openid" value="#{openid.id}"/>
    <h:commandButton value="Login" action="#{openid.login}"/>
</h:form>
```

- User returned to /openid.xhtml pseudo-view after login
- Navigation rules direct user
  - Transfer Open ID account to principal
  - Direct user to registration page
- Access Open ID using #{openid.validatedId}

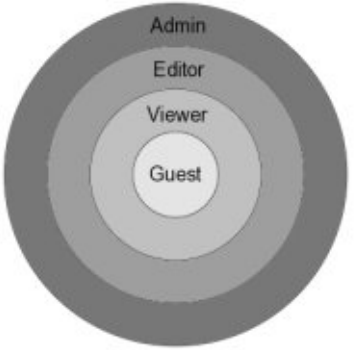
Open ID identity transfer



Authorization

Authorization styles

- **Binary**
  - Separates members from guests
- **Role-based**
  - Stereotypes users
  - Enforces privilege levels
- **Rule-based**
  - Declarative and contextual
- **Access control lists (ACLs)**
  - Protects object instances
  - Stored in data source



Binary authorization

- First line of defense
- Requires user to have identity
- identity component tracks “logged in” state

Java

```
if (identity.isLoggedIn()) {
    ...
}
```

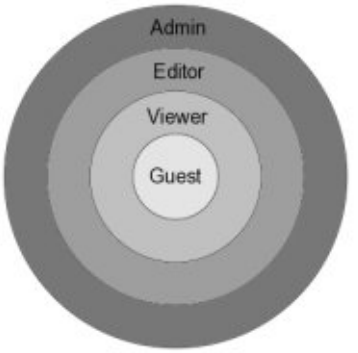
Seam page descriptor

```
<page view-id="/membersOnly.xhtml"
    login-required="true">
    ...
</page>
```

EL

```
<h:panelGroup rendered="#{identity.loggedIn}">
    Rate this post...
</h:panelGroup>
```


Role-based authorization



- Coarse-grained security
  - Primarily for partitioning application
- Roles assigned during authentication
  - identity.addRole()
  - @Roles
- Seam doesn't dictate naming convention

Java

```
if (identity.hasRole("admin")) {
    ...
}
```

JBoss EL

```
<s:link view="/admin/home.xhtml"
    rendered="#{identity.hasRole("admin")}"
    value="Admin Area"/>
```


Declarative restrictions

- JSF views

```
<page login-required="true"/>
```

```
<restrict>#{identity.hasRole("admin")}</restrict>
```

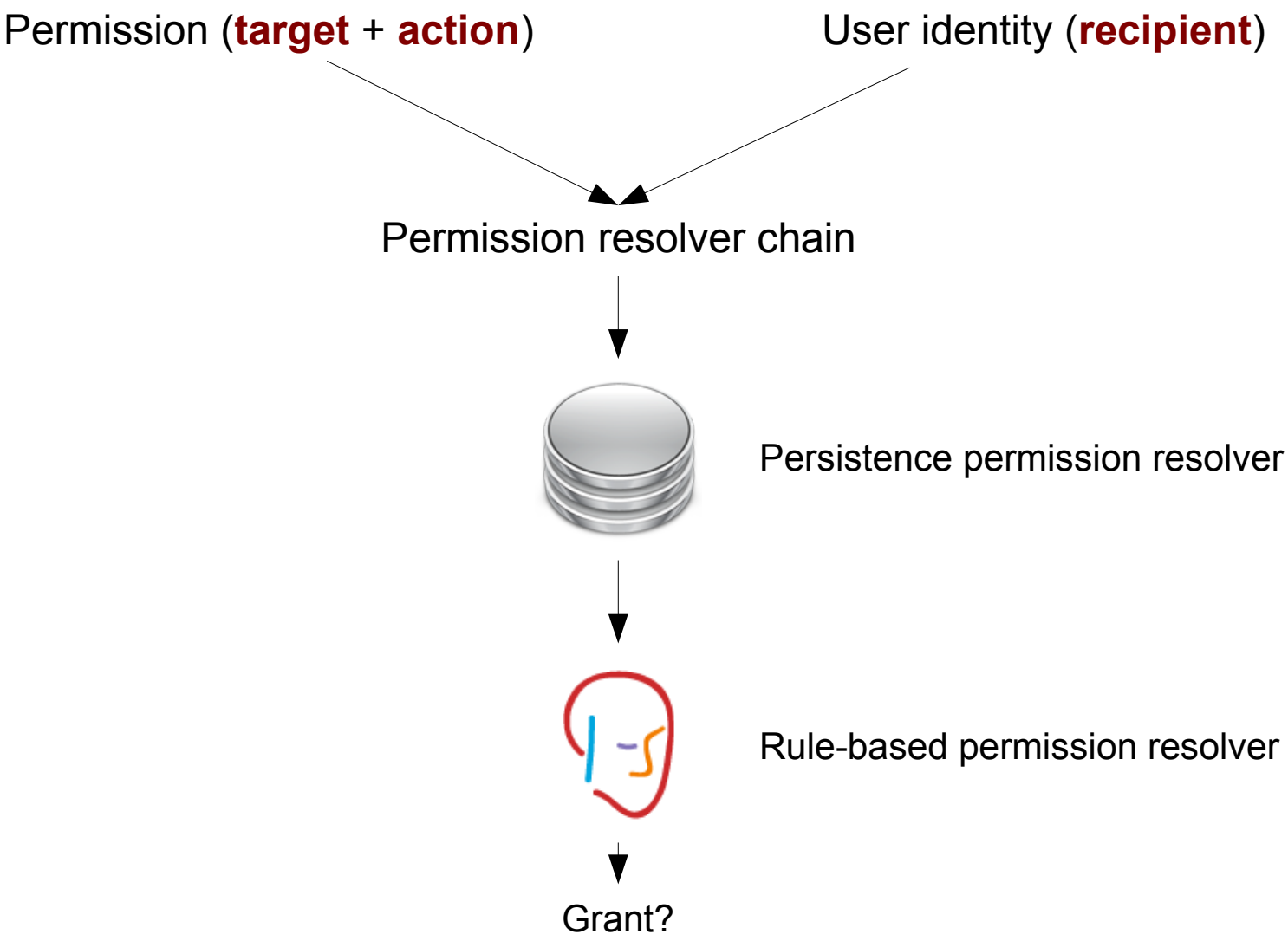
```
<h:panelGroup rendered="#{identity.loggedIn}">
    Rate this post...
</h:panelGroup>
```

- Classes and methods

```
public @Restrict void uploadDesign() { ... }
```

- Permission implied if no criteria
  - Target - object instance or view ID
  - Action - method name or JSF life cycle phase

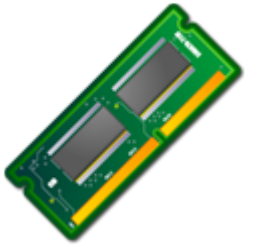
Resolving a permission



Rule-based security

- Based on Drools
- Major differentiator of Seam Security
- Rules are the raison d'être of security
  - You cannot enter the room with key
  - You cannot buy alcohol unless you are 21
  - You cannot fly if you have illegal weapons or 4oz of shampoo
  - You cannot cash check unless it’s endorsed
- Eliminates a lot of spaghetti business logic
  - Declarative and expressive
  - Hot swappable

Check your facts

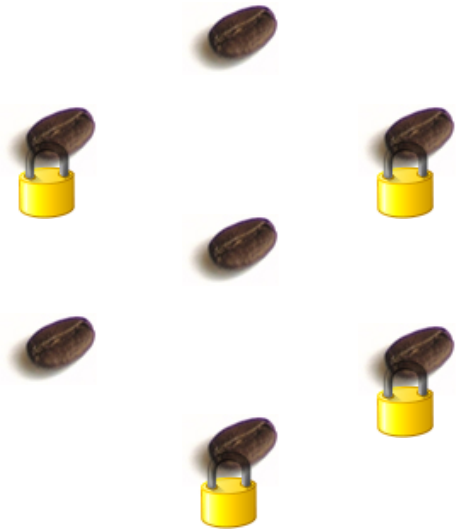


- The working memory contains facts
  - Facts are objects stored in a rules session
- You use facts to:
  - make assertions
  - perform operations when the rule is true
- Seam seeds working memory for security rules
  - PermissionCheck – the permission being requested
  - Principal – the security principal holding the username
  - Role – one or more roles assigned to the user
  - Authenticated user account (identity management)
  - Optional set of user-defined objects

Permission management

Access Control Lists (ACLs)

- Permission with a specific target
- Granted to a user, role or group
- Managed by the application
  - Typically persisted in a database
- Can be combined with rules
  - Conditional roles



SELECT * FROM MEMBER_PERMISSION;

PERMISSIONID	ACTION	DISCRIMINATOR	RECIPIENT	TARGET
2	view	role	admin	Design:2
4	view	role	client	Design:1
5	view	role	admin	Design:1

Managing permissions

- Permission
  - target
  - action
  - recipient (security principal)
- Select provider

```
<security:jpa-permission-store
    user-class="com.company.app.model.UserPermission"/>
```

- Manage
  - Use built-in permissionManager component
  - Provides list, grant and revoke operations

Demo: Permissions

King for a thread

- Elevated privileges for distinct operation
  - Self-registration on a web site

```
new RunAsOperation() {
   public void execute() {
       identityManager.createUser(username, password);
   }
}.addRole("admin").run();
```

- Alternatively, could use rule for this operation

Seam security take away

- *Easy* to adopt
- <configuration> is kept to a minimum
- Provides built-in security components
  - Declarative authentication
  - Identity and permissions management
- Has a myriad of authorization options
  - Binary, role-based, rule-based and ACLs
  - Combination of rules and ACLs is powerful
- **A security model that matures with your application**

Permission granted

- ✔ Ask questions
- ✔ Make comments
- ✔ Try out Seam Security

Resources

- Seam in Action, Manning 2008
  - Securing Seam Applications (Ch 11)
  - http://manning.com/dallen
- Seam project and news
  - http://seamframework.org
  - http://in.relation.to
- ACL Security in Seam, DZone article
  - http://java.dzone.com/articles/acl-security-in-seam
- Demo code
  - http://tinyurl.com/seamsecuritydemos