Apache Ki
(formerly JSecurity)

DevNexus - 2009
Introduction

- Jeremy Haile
  - Project Co-Founder
  - VP Product Development, WeTheCitizens
Agenda

• What is Apache Ki?

• Terminology

• Authentication, Authorization, Session Management, Cryptography

• Modules/Support

• Getting Started/Demo

• Questions
What is Ki?

• Simple, security API for Java
• Apache Incubator project
• Not tied to any framework or container - can run standalone
• Authentication, authorization, session management, cryptography
Why the new name?

- Trademark concerns over JSecurity
- Ki = Fortress (surrounded by moat) in Japanese
- Ki (pronounced key) works well with lock/key allusions
- Doesn’t tie us to Java in the future
- Super short package names and jar files!
Why another security framework?

- JAAS - confusing APIs, over-limiting, not good for application-level security
- Spring Security - tied to Spring Framework, uses unusual terminology
- None of the libraries support dynamic, runtime security changes out-of-the-box
- No libraries supported cross-domain sessions
Project Goals

• Simple to configure, yet powerful

• Easy to plugin custom authentication/authorization

• Clean object model making use of Java 5+ features (annotations, generics)

• Support dynamic, instance-level security out of the box

• Sessions not tied to web or EJBs
Terminology

- **Realm** - data access object for an application’s security components (users, roles, permissions)
- **Subject** - representation of an application user
- **Principals** - a subject’s identifying attributes (e.g. username, user ID, social security #)
Terminology (cont.)

• Credentials - secret information known only to a user used to verify identity

• Permission - atomic representation of ability to perform an action

• Role - a named collection of permissions
Ki Architecture

Subject

JSecurityManager

Authentication
Authorization
Session Management

Pluggable Realms (one or more)
- JDBC Realm
- LDAP Realm
- Custom Realm

Session DAO
- Session Cache

Cryptography API
Authentication

- The process of verifying someone is who they claim to be
- Most commonly involves a username and password
Authentication in Ki

• Most access to Ki is via Subject interface
• Web environments get by calling SecurityUtils.getSubject()
• Authenticate by calling Subject.login(...)  
• Throws exception if authentication fails
UsernamePasswordToken token =
new UsernamePasswordToken(username, password);

// Remember me is built in - just set to true:
token.setRememberMe( true );

try {
    Subject subject = SecurityUtils.getSubject();
    subject.login(token);

    // Variety of AuthenticationException subclasses can be caught
} catch (UnknownAccountException uae) { // User account not found
} catch (IncorrectCredentialsException ice) { // Password wrong
} catch (AuthenticationException e) { // Other auth error
}
Authorization

- The process of determining if someone can perform a specific action
- Ki supports using roles and/or permissions for authorization
- Ki doesn’t limit your data model
- Object or string based permissions
- Supports instance-level permissions
Permissions

• Most atomic element of authorization
• Not related to a user
• Ki supports two forms:
  • Object-based - extend Ki’s Permission interface
  • String-based - converted to object by PermissionResolver (default creates WildcardPermissions)
WildcardPermission

- Powerful string representation of permissions
- Simple - “clearCache” “generateReports”
- Multi-level - “users:edit” “users:delete”
- Instance level - “newsletter:send:13”
WildcardPermission
(cont.)

• Wildcards
  • “newsletter:*:13” (e.g. user allowed to perform any action for ID 13)

• Comma-lists
  • “newsletter:send:13,14,15” (e.g. user allowed to send for ID 13, 14, and 15)

• Can grant wildcards, but always check specific instance “newsletter:delete:13”
Roles

- Named collection of permissions
- Users associated with a role typically inherit the permissions associated with it
- Allows granting sets of permissions to users
- Ki doesn’t have a Role class - up to the Realm
Authorization in Ki

• Authorization mechanisms:
  • Programmatic (via Subject interface)
  • Declarative URL security (web.xml or jsecurity.ini)
  • Annotation-based method security
  • JSP tag-based
Programmatic Authorization

// Most common usage - obtain Subject from Thread Local
Subject subject = SecurityUtils.getSubject();

// Can programmatically check role
if( subject.hasRole( "manager" ) ) {
    // Logic for managers
}

// Or programmatically check permissions
if( subject.isPermitted( "message:delete" ) ) {
    // Allow user to delete a message
}

// Many other authorization methods on Subject
subject.checkPermission(...);
subject.hasAllRoles(...); etc.
// Permissions can be checked using object-based permissions
PrinterPermission perm = new PrinterPermission("hplaserjet");
if( subject.isPermitted( perm ) ) {
    ...
}

// or string-based permissions
if( subject.isPermitted("printer:hplaserjet") ) {
    ...
}
URL Authorization

INI web configuration (more on this later)

[urlsl]
# Allow anonymous access
/s/signup=anon

# Require roles for access
/s/manageUsers=roles[admin]

# Require permissions for access
/s/manageUsers=perms[user:manage]

# Require user to be authenticated
/s/**=authc
// Will automatically throw an AuthorizationException if caller
// doesn’t have required role
@RequiresRoles("manager")
public String editUser(...) {
}

// Will automatically throw an AuthorizationException if caller
// doesn’t have required permission
@RequiresPermissions("user:edit")
public String editUser(...) {
}
JSP Tag Authorization

<%@ taglib prefix="jsec" uri="http://www.jsecurity.org/tags" %>

<%-- Check permissions --%>
<jsec:hasPermission name="user:manage">....</jsec:hasPermission>
<jsec:lacksPermission name="user:manage">....</jsec:lacksPermission>

<%-- Check roles --%>
<jsec:hasRole name="admin">....</jsec:hasRole>
<jsec:lacksRole name="admin">....</jsec:lacksRole>

<%-- Some other tags... --%>
<jsec:authenticated>...</jsec:authenticated>
<jsec:user>...</jsec:user>
<jsec:guest>...</jsec:guest>
Realms

• Encapsulates both authentication and authorization

• Built in realms for JDBC, LDAP, property files, etc.

• Or create custom Realm for your data model

• Usually custom Realms will extend AuthorizingRealm
// From Spring-Hibernate sample app
public class SampleRealm extends AuthorizingRealm {

    public SampleRealm() {
        setCredentialsMatcher(new Sha256CredentialsMatcher());
    }

    // Superclass will compare credentials using an Sha256 hash as specified above
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken authcToken) {
        UsernamePasswordToken token = (UsernamePasswordToken) authcToken;

        User user = userDAO.findUser(token.getUsername());
        if (user != null) {
            return new SimpleAuthenticationInfo(user.getId(), user.getPassword(), getName());
        } else {
            return null;
        }
    }
}
Custom Realm Example (authorization)

// From Spring-Hibernate sample app

// Retrieve user info from DAO and return AuthorizationInfo
protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
    Long userId = (Long) principals.fromRealm(getName()).iterator().next();
    User user = userDAO.getUser(userId);
    if( user != null ) {
        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();
        for( Role role : user.getRoles() ) {
            info.addRole(role.getName());
            info.addStringPermissions( role.getPermissions() );
        }
        return info;
    } else {
        return null;
    }
}
Session Management

- Supports heterogeneous clients
- Does not require HTTP environment (works with server-side service, Swing client, applets, etc.)
- POJO-based (IoC friendly)
- Can be used for Single-Sign On
Session Management (cont.)

- Temporal-based record (start, stop, last access)
- Can transparently be used to back HttpSession
- Session event listener support
- Supports inactivity/expiration
- Allows setting attributes (similar to HttpSession)
Ki Sessions

// Access sessions via the subject - session created automatically
Session session = subject.getSession();

// Session only returned if it already exists
Session session = subject.getSession(false);

// Can perform a variety of session operations
session.setAttribute("userId", user.getId());
session.getAttribute("userId");
session.getLastAccessTime();
Session Data

- Can configure to use any storage mechanism (file system, memory, distributed cache, JDBC)
- Allows easy ability to cluster (Terracotta, GigaSpaces, etc.)
Cryptography

- Interface driven, POJO-based
- Simplified wrapper of complicated JCE infrastructure
- Easy to understand API (ciphers, hashes, etc.)
Ciphers

- A cipher is a cryptographic algorithm that encrypts and decrypts data using public and/or private keys
  
  - Symmetric cipher - uses the same key to encrypt and decrypt
  
  - Asymmetric cipher - uses one key to encrypt and another to decrypt
Ki Ciphers

```java
public interface Cipher {
    byte[] encrypt(byte[] raw, byte[] encryptionKey);
    byte[] decrypt(byte[] encrypted, byte[] decryptionKey);
}
```

- Default implementations exist (BlowfishCipher, etc.)
- Encrypt user identities, such as remember me, cookies, etc.
A cryptographic hash is a one-way, irreversible conversion of an input source. Commonly used for passwords. Can be used on anything (files, streams, etc.).
Ki Hashes

//some examples:
new Md5Hash("blah").toHex();

//File MD5 Hash value for checksum:
new MD5Hash(aFile).toHex();

//store a password, but not in raw form:
new Sha256(aPassword).toBase64();

- Cleaner OO-API compared to JDK MessageDigest and Commons Digest
- Built-in hex/base64 conversion
- Built-in support for salts/repeated hashing
Modules/Support

• Packaged as modules (jsecurity-core, jsecurity-spring, jsecurity-ehcache, ...)
• Web module for integration with web applications
• Spring Framework integration
• Grails plugin
• Guice, JBoss support on the way (usable now, but not officially supported)
Web Integration

<!-- Excerpt from the Spring-Hibernate sample app -->

<filter>
    <filter-name>JSecurityFilter</filter-name>

    <!-- The below class is for Spring/web - for non-Spring change to just JSecurityFilter -->
    <filter-class>org.jsecurity.spring.SpringJSecurityFilter</filter-class>
    <init-param>
        <param-name>config</param-name>
        <param-value>
            [filters]
            jsecurity.loginUrl = /s/login
            jsecurity.unauthorizedUrl = /unauthorized.jsp
            authc.successUrl = /s/home

            [urls]
            /s/signup=anon
            /s/manageUsers=perms[user:manage]
            /s/**=authc
        </param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>JSecurityFilter</filter-name>
    <url-pattern>/s/*</url-pattern>
</filter-mapping>
<bean id="securityManager" class="org.jsecurity.web.DefaultWebSecurityManager">

<!-- If you have multiple realms, use the 'realms' property instead. -->
<property name="realm" ref="sampleRealm"/>

<!-- To use JSecurity sessions, set the session mode -->
<property name="sessionMode" value="jsecurity"/>
</bean>

<!-- Post processor that automatically invokes init() and destroy() methods -->
<bean id="lifecycleBeanPostProcessor"
     class="org.jsecurity.spring.LifecycleBeanPostProcessor"/>
Samples/Demo

- Spring-Hibernate Sample App
- Typical three-tier web app
- Uses Spring 2.5 and JPA annotations
- Demonstrates authentication, authorization, logout, annotations, JSP tags, etc.
- Other samples include standalone and web site/WebStart shared sessions
Status

• Latest release: JSecurity 0.9
• 1.0 release in the works
  • Packages/docs/code renamed to Ki
  • Assumed Identity (run-as) support
  • Authentication caching
  • More built-in Realms (OpenID, Atlassian Crowd)
• Other enhancements (tracked in JIRA)
Questions

Website: http://www.jsecurity.org
Mailing Lists: jsecurity-user@incubator.apache.org
             jsecurity-dev@incubator.apache.org