# Hacking the Grails Spring Security Plugins

*Burt Beckwith, SpringSource*

springsource
A division of vmware

CONFIDENTIAL

# web.xml

# web.xml

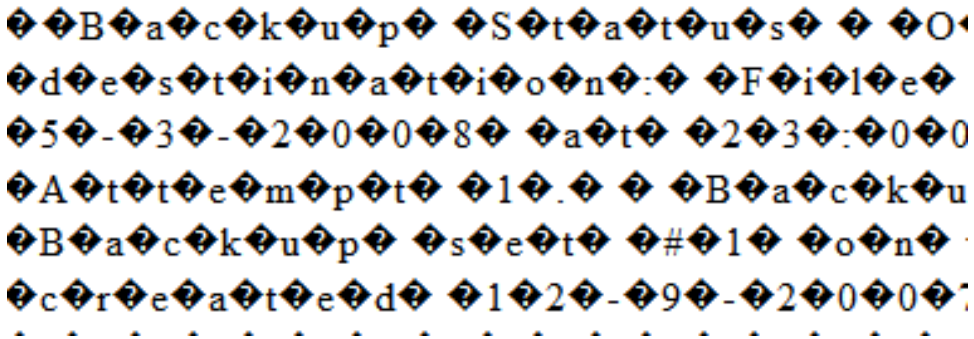
- Spring Security is implemented as a filter chain, so the ***<filter-mapping>*** elements are the important ones:
  - charEncodingFilter
  - hiddenHttpMethod
  - grailsWebRequest
  - springSecurityFilterChain
  - reloadFilter
  - sitemesh
  - urlMapping

# web.xml

- Spring Security is implemented as a filter chain, so the ***<filter-mapping>*** elements are the important ones:
  - charEncodingFilter
  - hiddenHttpMethod
  - grailsWebRequest
  - **springSecurityFilterChain**
  - reloadFilter
  - sitemesh
  - urlMapping

# web.xml - charEncodingFilter

- org.springframework.web.filter.DelegatingFilterProxy
- Uses the "**`characterEncodingFilter`**" bean (**`org.springframework.web.filter.CharacterEncodingFilter`**) defined in **`applicationContext.xml`** (the “parent” context)
- ***`request.setCharacterEncoding("utf-8");`***
- ***`response.setCharacterEncoding("utf-8");`***



# web.xml - hiddenHttpMethod

- org.codehaus.groovy.grails.web.filters.HiddenHttpMethodFilter

```
String httpMethod = getHttpMethodOverride(request);
filterChain.doFilter(
   new HttpMethodRequestWrapper(httpMethod, request),
   response);
```

- <g:form controller="book" method="DELETE">

- See section “13.1 REST” in the docs

# web.xml - grailsWebRequest

- org.codehaus.groovy.grails.web.servlet.mvc.GrailsWebRequestFilter

```
LocaleContextHolder.setLocale(request.getLocale());
GrailsWebRequest webRequest = new GrailsWebRequest(
       request, response, getServletContext());
configureParameterCreationListeners(webRequest);

try {
   WebUtils.storeGrailsWebRequest(webRequest);

   // Set the flash scope instance to its next state
   FlashScope fs = webRequest.getAttributes().getFlashScope(request);
   fs.next();

   filterChain.doFilter(request, response);
}
finally {
   webRequest.requestCompleted();
   WebUtils.clearGrailsWebRequest();
   LocaleContextHolder.setLocale(null);
}
```

# web.xml - springSecurityFilterChain

- org.springframework.web.filter.DelegatingFilterProxy
- Uses the "**`springSecurityFilterChain`**" bean defined by the plugin (**`org.springframework.security.web.FilterChainProxy`**)

# web.xml - springSecurityFilterChain

- Typical filter beans:
  - securityContextPersistenceFilter
  - logoutFilter
  - authenticationProcessingFilter
  - securityContextHolderAwareRequestFilter
  - rememberMeAuthenticationFilter
  - anonymousAuthenticationFilter
  - exceptionTranslationFilter
  - filterInvocationInterceptor

# web.xml - reloadFilter

- org.codehaus.groovy.grails.web.servlet.filter.GrailsReloadServletFilter
- No longer used in 2.0+
- “Copies resources from the source on content change and manages reloading if necessary.”

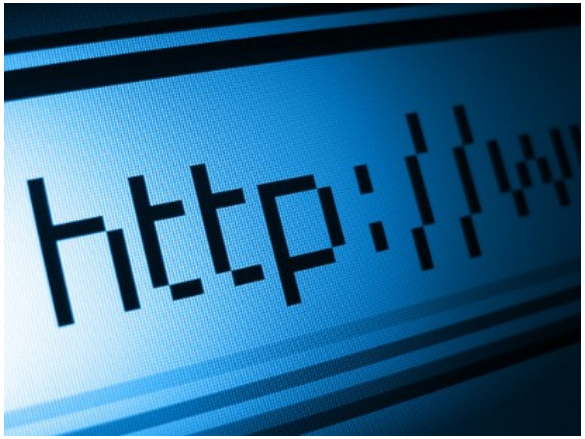
```
GrailsApplication application = (GrailsApplication)context.getBean(
      GrailsApplication.APPLICATION_ID);
if (!application.isInitialised()) {
   application.rebuild();
   GrailsRuntimeConfigurator config = new GrailsRuntimeConfigurator(
      application, parent);
   config.reconfigure(context, getServletContext(), true);
}
```

# web.xml - sitemesh

- org.codehaus.groovy.grails.web.sitemesh.GrailsPageFilter
- Subclass of `com.opensymphony.sitemesh.webapp.SiteMeshFilter`
- Renders Sitemesh templates/layouts

# web.xml - urlMapping

- org.codehaus.groovy.grails.web.mapping.filter.UrlMappingsFilter
- Matches requested url to the correct controller/action/view/error code
- Based on mappings in ***`grails-app/conf/UrlMappings.groovy`***



# springSecurityFilterChain

# springSecurityFilterChain - securityContextPersistenceFilter

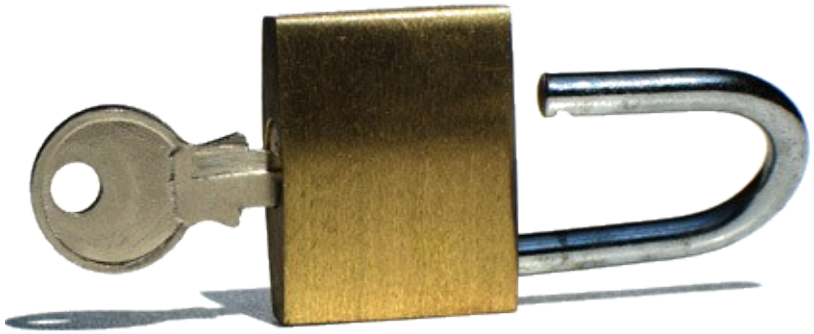
- org.springframework.security.web.context.SecurityContextPersistenceFilter
- Retrieves the ***`SecurityContext`*** from the HTTP session (under the ***`"SPRING_SECURITY_CONTEXT"`*** key) or creates a new empty one
- Stores the context in the holder: ***`SecurityContextHolder.setContext()`***
- Removes the context after request

# springSecurityFilterChain - logoutFilter

- org.codehaus.groovy.grails.plugins.springsecurity.MutableLogoutFilter
- If uri is "/**j_spring_security_logout**" calls ***handler.logout(request, response, auth)*** for each **o.s.s.web.authentication.logout.LogoutHandler** and redirects to post-logout url (by default "**/**")

# springSecurityFilterChain - authenticationProcessingFilter

- org.codehaus.groovy.grails.plugins.springsecurity.RequestHolderAuthenticationFilter (extends o.s.s.web.authentication.UsernamePasswordAuthenticationFilter)
- Sets the request and response in the ***SecurityRequestHolder ThreadLocal*** fields (custom plugin functionality)
- If uri is **"/j_spring_security_check"** calls ***attemptAuthentication()***



# springSecurityFilterChain - securityContextHolderAwareRequestFilter

- o.s.s.web.servletapi.SecurityContextHolderAwareRequestFilter
- Configures a request wrapper which implements the servlet API security methods (***getRemoteUser***, ***getUserPrincipal***, ***isUserInRole***)

```
chain.doFilter(new SecurityContextHolderAwareRequestWrapper(
    request, rolePrefix),
    response);
```

# springSecurityFilterChain - rememberMeAuthenticationFilter

- o.s.s.web.authentication.rememberme.RememberMeAuthenticationFilter
- If not logged in, attempt to login from the remember-me cookie

# springSecurityFilterChain - anonymousAuthenticationFilter

- o.s.s.web.authentication.AnonymousAuthenticationFilter
- If not logged in creates an ***`AnonymousAuthenticationToken`*** and registers it as the ***`Authentication`*** in the ***`SecurityContext`***

https://secure.flickr.com/photos/gaelx/5445598436/

# springSecurityFilterChain - exceptionTranslationFilter

- o.s.s.web.access.ExceptionTranslationFilter
- Handles ***AccessDeniedException*** and ***AuthenticationException***
- For ***AuthenticationException*** will invoke the ***authenticationEntryPoint***
- For ***AccessDeniedException*** invoke the ***authenticationEntryPoint*** if anonymous, otherwise delegate to **o.s.s.web.access.AccessDeniedHandler**

# springSecurityFilterChain - filterInvocationInterceptor

- o.s.s.web.access.intercept.FilterSecurityInterceptor
- Determines the `o.s.s.access.SecurityConfig` collection for the request from ***`FilterInvocationSecurityMetadataSource`*** (***`AnnotationFilterInvocationDefinition`***, ***`InterceptUrlMapFilterInvocationDefinition`***, or ***`RequestmapFilterInvocationDefinition`***)

# springSecurityFilterChain - filterInvocationInterceptor

- Delegates to an ***`AccessDecisionManager`***

  (`org.codehaus.groovy.grails.plugins.springsecurity.`

  **`AuthenticatedVetoableDecisionManager`**) to call each registered

  `o.s.s.access.`**`AccessDecisionVoter`**

# springSecurityFilterChain - filterInvocationInterceptor

- `o.s.s.access.vote.AuthenticatedVoter` works with

  "`IS_AUTHENTICATED_FULLY`", "`IS_AUTHENTICATED_REMEMBERED`",

  and "`IS_AUTHENTICATED_ANONYMOUSLY`"

# springSecurityFilterChain - filterInvocationInterceptor

- `o.s.s.access.vote.RoleHierarchyVoter` finds the ***`GrantedAuthority`*** instances from the ***`Authentication`***, checks for matches with required roles

# springSecurityFilterChain - filterInvocationInterceptor

- `org.codehaus.groovy.grails.plugins.springsecurity.`

  **`WebExpressionVoter`** looks for a

  ***`WebExpressionConfigAttribute`*** and evaluates its expression if

  found

# springSecurityFilterChain - filterInvocationInterceptor

```
if (denyCount > 0) {
    throw new AccessDeniedException("Access is denied");
}
```

- this is caught by the ***exceptionTranslationFilter***
- If the ***Authentication*** is anonymous, stores a ***SavedRequest*** in the HTTP session and calls ***authenticationEntryPoint.commence( request, response, reason)***
- The ***authenticationEntryPoint*** is an `org.codehaus.groovy.grails.plugins.springsecurity.AjaxAwareAuthenticationEntryPoint`
- This issues a redirect to the login page, by default ***/login/auth***

# Customizing the Plugin

# Customizing the Plugin

Traditional Spring Security uses XML with namespaced beans:

```
<beans:beans
     xmlns="http://www.springframework.org/schema/security"
     ...>

  <http auto-config="true">
    <intercept-url pattern="/admin/**" access="ROLE_ADMIN" />
  </http>

  <authentication-manager>
    <authentication-provider>
      <jdbc-user-service data-source-ref="dataSource" />
    </authentication-provider>
  </authentication-manager>

</beans:beans>
```

# Customizing the Plugin

- Simple; not always clear how to customize; many options aren't exposed
- In contrast, the plugin explicitly defines *every* bean
- See ***`SpringSecurityCoreGrailsPlugin.groovy`*** for the details

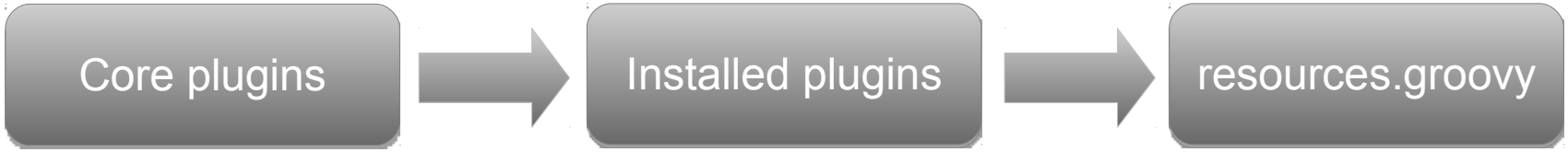
# Overriding Beans

# Overriding Beans

accessDecisionManager
accessDeniedHandler
anonymousAuthenticationFilter
anonymousAuthenticationProvider
authenticatedVoter
authenticationDetailsSource
authenticationEntryPoint
authenticationEventPublisher
authenticationFailureHandler
authenticationManager
authenticationProcessingFilter
authenticationSuccessHandler
authenticationTrustResolver
authenticationUserDetailsService
daoAuthenticationProvider
exceptionTranslationFilter
filterInvocationInterceptor
logoutFilter
logoutHandlers
logoutSuccessHandler
objectDefinitionSource
passwordEncoder
portMapper
portResolver
postAuthenticationChecks
preAuthenticationChecks
redirectStrategy
rememberMeAuthenticationFilter
rememberMeAuthenticationProvider
rememberMeServices
requestCache
roleHierarchy
roleVoter
runAsManager
saltSource
securityContextHolderAwareRequestFilter
securityContextLogoutHandler
securityContextPersistenceFilter
securityEventListener
sessionAuthenticationStrategy
springSecurityFilterChain
tokenRepository
userCache
userDetailsService
webExpressionHandler
webExpressionVoter
webInvocationPrivilegeEvaluator

# Overriding Beans

- Building the Spring ***`ApplicationContext`***



# Overriding Beans

- Gross oversimplifications:
  - The **`ApplicationContext`** is like a **`Map`**; the keys are bean names and the values are bean instances
  - Defining a bean with the same name as an earlier bean replaces the previous, just like ***`Map.put(key, value)`*** does

# Overriding Beans

- To change how a bean works:
  - Subclass the current class
  - Or subclass a similar class that's closer to what you need
  - Or implement the interface directly
  - Then register yours in ***grails-app/conf/spring/resources.groovy***

# Overriding Beans - resources.groovy

```
beans = {

   saltSource(com.foo.bar.MySaltSource) {
      // bean properties
   }

   userDetailsService(com.foo.bar.MyUserDetailsService) {
      // bean properties
   }

   passwordEncoder(com.foo.bar.MyPasswordEncoder) {
      // bean properties
   }

}
```

# Customizing Bean Properties

# Customizing Bean Properties

- In addition to declaring every bean, nearly all properties are configured from default values in the plugin's

  ***grails-app/conf/DefaultSecurityConfig.groovy***

- DO NOT EDIT ***DefaultSecurityConfig.groovy*** 

# Customizing Bean Properties

- All properties can be overridden in app's ***`Config.groovy`***
  - Be sure to add the ***`grails.plugins.springsecurity`*** prefix

- Don't replace a bean if all you need to change is one or more properties

# Customizing Bean Properties

active
adh.ajaxErrorPage
adh.errorPage
ajaxHeader
anon.key
anon.userAttribute
apf.allowSessionCreation
apf.continueChainBeforeSuccessfulAuthentication
apf.filterProcessesUrl
apf.passwordParameter
apf.postOnly
apf.usernameParameter
atr.anonymousClass
atr.rememberMeClass
auth.ajaxLoginFormUrl
auth.forceHttps
auth.loginFormUrl
auth.useForward
authenticationDetails.authClass
authority.className
authority.nameField
basic.realmName
cacheUsers
controllerAnnotations.lowercase
controllerAnnotations.matcher
controllerAnnotations.staticRules
dao.hideUserNotFoundExceptions
dao.reflectionSaltSourceProperty
digest.createAuthenticatedToken
digest.key
digest.nonceValiditySeconds
digest.passwordAlreadyEncoded
digest.realmName
digest.useCleartextPasswords
failureHandler.ajaxAuthFailUrl
failureHandler.defaultFailureUrl
failureHandler.exceptionMappings
failureHandler.useForward
filterChain.stripQueryStringFromUrls
interceptUrlMap
ipRestrictions
logout.afterLogoutUrl
logout.filterProcessesUrl
logout.handlerNames
password.algorithm
password.bcrypt.logrounds
password.encodeHashAsBase64
portMapper.httpPort
portMapper.httpsPort
providerManager.eraseCredentialsAfterAuthentication
providerNames
redirectStrategy.contextRelative
registerLoggerListener
rejectIfNoRule
rememberMe.alwaysRemember
rememberMe.cookieName
rememberMe.key
rememberMe.parameter
rememberMe.persistent
rememberMe.persistentToken.domainClassName
rememberMe.persistentToken.seriesLength
rememberMe.persistentToken.tokenLength
rememberMe.tokenValiditySeconds
rememberMe.useSecureCookie
requestCache.createSession
requestCache.onlyOnGet
requestMap.className
requestMap.configAttributeField
requestMap.urlField
roleHierarchy
secureChannel.definition
securityConfigType
sessionFixationPrevention.alwaysCreateSession
sessionFixationPrevention.migrate
successHandler.ajaxSuccessUrl
successHandler.alwaysUseDefault
successHandler.defaultTargetUrl
successHandler.targetUrlParameter
successHandler.useReferer
switchUser.exitUserUrl
switchUser.switchFailureUrl
switchUser.switchUserUrl
switchUser.targetUrl
useBasicAuth
useDigestAuth
useHttpSessionEventPublisher
userLookup.accountExpiredPropertyName
userLookup.accountLockedPropertyName
userLookup.authoritiesPropertyName
userLookup.authorityJoinClassName
userLookup.enabledPropertyName
userLookup.passwordExpiredPropertyName
userLookup.passwordPropertyName
userLookup.userDomainClassName
userLookup.usernamePropertyName
useSecurityEventListener
useSessionFixationPrevention
useSwitchUserFilter
useX509
voterNames
x509.checkForPrincipalChanges
x509.continueFilterChainOnUnsuccessfulAuthentication
x509.invalidateSessionOnPrincipalChange
x509.subjectDnRegex
x509.throwExceptionWhenTokenRejected

# Customizing Bean Properties

grails-app/conf/Config.groovy:

```
grails.plugins.springsecurity.userLookup.userDomainClassName = '…'

grails.plugins.springsecurity.userLookup.authorityJoinClassName = '…'

grails.plugins.springsecurity.authority.className = '…'

grails.plugins.springsecurity.auth.loginFormUrl = '/log-in'

grails.plugins.springsecurity.apf.filterProcessesUrl = '/authenticate'

grails.plugins.springsecurity.logout.afterLogoutUrl = '/home'

grails.plugins.springsecurity.userLookup.usernamePropertyName = 'email'
```

# Customizing Bean Properties

- Can also configure beans in ***BootStrap.groovy***, e.g. to avoid redefining a whole bean in ***resources.groovy*** just to change one dependency:

```
class BootStrap {

   def authenticationProcessingFilter
   def myAuthenticationFailureHandler

   def init = { servletContext ->
      authenticationProcessingFilter.authenticationFailureHandler =
            myAuthenticationFailureHandler
   }

}
```

# Custom UserDetailsService

# Custom UserDetailsService

- Very common customization
- Documented in the plugin docs in section

  “11 Custom UserDetailsService”

# Custom UserDetailsService

- General workflow:
  - Create a custom `o.s.s.core.userdetails.`**`UserDetailsService`** (`o.c.g.g.p.s.`**`GrailsUserDetailsService`**) implementation
    - Directly implement the interface (best)
    - or extend `org.codehaus.groovy.grails.plugins.springsecurity.`**`GormUserDetailsService`** (ok, but usually cleaner to implement the interface)

# Custom UserDetailsService

- General workflow (cont.):
  - Create a custom implementation of `o.s.s.core.userdetails.UserDetails`
    - Extend `org.codehaus.groovy.grails.plugins.springsecurity.GrailsUser`
    - or extend `o.s.s.core.userdetails.User`
    - or directly implement the interface

# Custom UserDetailsService

- General workflow (cont.):
  - Register the bean override in ***grails-app/conf/spring/resources.groovy***:

```
import com.mycompany.myapp.MyUserDetailsService

beans = {
   userDetailsService(MyUserDetailsService)
}
```

# Custom AuthenticationProvider

# Custom AuthenticationProvider

- General workflow:
  - Create a custom `o.s.s.authentication.AuthenticationProvider` implementation
    - Extend one that's similar, e.g.

      `o.s.s.authentication.dao.DaoAuthenticationProvider`

    - or directly implement the interface

# Custom AuthenticationProvider

- General workflow (cont.):
  - You'll probably want a custom `o.s.s.core.`**`Authentication`**
    - Extend an existing implementation, e.g.

      `o.s.s.authentication.`**`UsernamePasswordAuthenticationToken`**

    - or directly implement the interface

# Custom AuthenticationProvider

- General workflow (cont.):
  - If you're using a custom ***`Authentication`***, you'll need a filter to create it
    - Create a custom `javax.servlet.Filter` implementation
    - Best to extend `org.springframework.web.filter.`**`GenericFilterBean`**
    - or one that's similar, e.g.

      `o.s.s.web.authentication.`**`UsernamePasswordAuthenticationFilter`**
    - Or directly implement the interface

# Custom AuthenticationProvider

- Registering the custom classes
  - If you're replacing functionality, register bean overrides in ***`resources.groovy`***
  - If you're adding an alternate authentication option (e.g. for only some URLs)
    - Register the beans in ***`resources.groovy`***
    - Register the provider as described in section “10 Authentication Providers” of the docs
    - Register the filter in its position as described in section “16 Filters” of the docs

# Custom Post-logout Behavior

# Custom Post-logout Behavior

- Recall that logout is processed by **`MutableLogoutFilter`** after request for **`/j_spring_security_logout`**
- ***`handler.logout(request, response, auth)`*** is called for each ***`LogoutHandler`***
- then redirect to post-logout url (by default *"/"*)

# Custom Post-logout Behavior

- The default **`LogoutHandler`** implementations are
  - **o.s.s.web.authentication.rememberme.`TokenBasedRememberMeServices`**
    - Deletes the remember-me cookie
  - **o.s.s.web.authentication.logout.`SecurityContextLogoutHandler`**
    - Invalidates the ***`HttpSession`***
    - Removes the ***`SecurityContext`*** from the ***`SecurityContextHolder`***

# Custom Post-logout Behavior

- Redirect is triggered by
  - ***logoutSuccessHandler.onLogoutSuccess(request, response, auth)***
  - ***LogoutSuccessHandler*** is an instance of `o.s.s.web.authentication.logout.`**`SimpleUrlLogoutSuccessHandler`**

# Custom Post-logout Behavior

- To add logic to dynamically determine redirect url:
  - Subclass ***`SimpleUrlLogoutSuccessHandler`***
  - Since the ***`Authentication`*** isn't available in ***`determineTargetUrl`***, override ***`onLogoutSuccess`*** and set it in a ***`ThreadLocal`***

# Custom Post-logout Behavior

- To add logic to dynamically determine redirect url (cont.):

```
private static final ThreadLocal<Authentication> AUTH_HOLDER =
     new ThreadLocal<Authentication>()
…

void onLogoutSuccess(...) throws ... {
   AUTH_HOLDER.set authentication
   try {
      super.handle(request, response, authentication)
   }
   finally {
      AUTH_HOLDER.remove()
   }
}
```

# Custom Post-logout Behavior

- To add logic to dynamically determine redirect url (cont.):
  - Override ***determineTargetUrl***

```
protected String determineTargetUrl(...) {
   Authentication auth = AUTH_HOLDER.get()

   String url = super.determineTargetUrl(request, response)

   if (auth instanceof OrganizationAuthentication) {
      OrganizationAuthentication authentication = auth
      url = ...
   }

   url
}
```

# Custom Post-logout Behavior

- To add logic to dynamically determine redirect url (cont.):
  - Redefine the ***logoutSuccessHandler*** bean in ***resources.groovy***

```
import com.mycompany.myapp.CustomLogoutSuccessHandler
import o.c.g.g.plugins.springsecurity.SpringSecurityUtils

beans = {

   def conf = SpringSecurityUtils.securityConfig

   logoutSuccessHandler(CustomLogoutSuccessHandler) {
      redirectStrategy = ref('redirectStrategy')
      defaultTargetUrl = conf.logout.afterLogoutUrl
   }
}
```

# Customization Overview

# Customization Overview

- Subclass or replace filters in the chain:
  - ***SecurityContextPersistenceFilter***
  - ***MutableLogoutFilter***
  - ***RequestHolderAuthenticationFilter***
    ***(UsernamePasswordAuthenticationFilter)***
  - ***SecurityContextHolderAwareRequestFilter***
  - ***RememberMeAuthenticationFilter***
  - ***AnonymousAuthenticationFilter***
  - ***ExceptionTranslationFilter***
  - ***FilterSecurityInterceptor***

# Customization Overview (cont)

- Remove filter(s) from the chain
  - ***Not recommended***
- Add filter(s) to the chain
- Add/remove/replace ***LogoutHandler***(***s***)
- Add/remove/replace ***AccessDecisionVoter***(***s***)
- Add/remove/replace ***AuthenticationProvider***(***s***)

# Customization Overview (cont)

- Customize a filter or ***`AuthenticationProvider`***'s dependency
  - e.g. custom ***`UserDetailsService`***
- Don't write code if you can customize properties or ***`BootStrap.groovy`***
- Read the Spring Security documentation
  - Print the PDF and read it on your commute
- Ask for help on the Grails User mailing list

# Thank you