OCI | HOME TO GRAILS

Securing your Grails application:

A survey/layout of the security plugin space

Colin Harrington

GR8Conf US 2016

whoami > Colin Harrington

- Grails team at OCI
- @ColinHarrington
- harringtonc@ociweb.com

OCI | HOME TO GRAILS

Grails is a **powerful** web framework, for the Java platform aimed at multiplying developers' productivity thanks to a Convention-over-Configuration, sensible defaults and opinionated APIs. It integrates smoothly with the JVM, allowing you to be immediately productive whilst providing powerful features, including integrated ORM, **Domain-Specific Languages**, runtime and compile-time **meta-programming** and **Asynchronous** programming.

Spring Security

Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.

History

- Acegi
- Spring-security-core
- Grails 3

Burt Beckwith

Spring Security Core

The Spring Security plugin simplifies the integration of Spring Security into Grails applications. The plugin provides sensible defaults with many configuration options for customization. Nearly everything is configurable or replaceable in the plugin and in Spring Security itself, which makes extensive use of interfaces.

grails-spring-security-core

- https://bintray.com/grails/plugins/spring-security-core
- https://github.com/grails-plugins/grails-spring-security-core
- https://grails-plugins.github.io/grails-spring-security-core/

Grails Artefacts

- SpringSecurityService
- SecurityTagLib
- LoginController & LogoutController

Highly Configurable

Sensible defaults

@Secured annotation

```
@Secured(value=["hasRole('ROLE_ADMIN')"]
def someMethod() { ... }
```

```
WebSecurityExpressionRoot delegate...

@Secured(closure = {
   assert request
   assert ctx
   authentication.name == 'admin1'
})
def someMethod() { ... }
```

Authentication Mechanisms

- Basic & Digest
- Cert - X.509
- Remember-me cookies
- Username/Password

FilterChain

```
grails.plugin.springsecurity.filterChain.filterNames = [
    'securityContextPersistenceFilter', 'logoutFilter',
    'authenticationProcessingFilter', 'myCustomProcessingFilter',
    'rememberMeAuthenticationFilter', 'anonymousAuthenticationFilter',
    'exceptionTranslationFilter', 'filterInvocationInterceptor'
]
```

chainMap

```
grails.plugin.springsecurity.filterChain.chainMap = [
   [pattern: '/urlpattern1/**', filters: 'filter1,filter2,filter3,filter4'],
   [pattern: '/urlpattern2/**', filters: 'filter1,filter3,filter5'],
   [pattern: '/**',             filters: 'JOINED_FILTERS']
]
```

grails-spring-security-core

Plugins

- grails-security-bridge
- grails-spring-security-acl
- grails-spring-security-appinfo
- grails-spring-security-cas
- grails-spring-security-core
- grails-spring-security-facebook
- grails-spring-security-jaxrs
- grails-spring-security-kerberos
- grails-spring-security-ldap
- grails-spring-security-oauth2
- grails-spring-security-oauth2-facebook
- grails-spring-security-oauth2-google
- grails-spring-security-oauth2-provider
- grails-spring-security-rest
- grails-spring-security-shiro
- grails-spring-security-ui

Authentication Providers

- LDAP
- CAS
- Kerberos

Tools

- appinfo
- ui

Enhancement

- acl
- jaxrs

Token/Stateless

- rest

oauth2

- oauth2-provider
- oath2
  - oauth2-facebook
  - oauth2-google

Adapter

- shiro
- security-bridge

UserDetails

```
public interface UserDetails extends Serializable {
        Collection<? extends GrantedAuthority> getAuthorities();
        String getPassword();
        String getUsername();
        boolean isAccountNonExpired();
        boolean isAccountNonLocked();
        boolean isCredentialsNonExpired();
        boolean isEnabled();
}
```

UserDetailsService

```
public interface UserDetailsService {
        UserDetails loadUserByUsername(String username)
            throws UsernameNotFoundException;
}
```

AuthenticationProvider

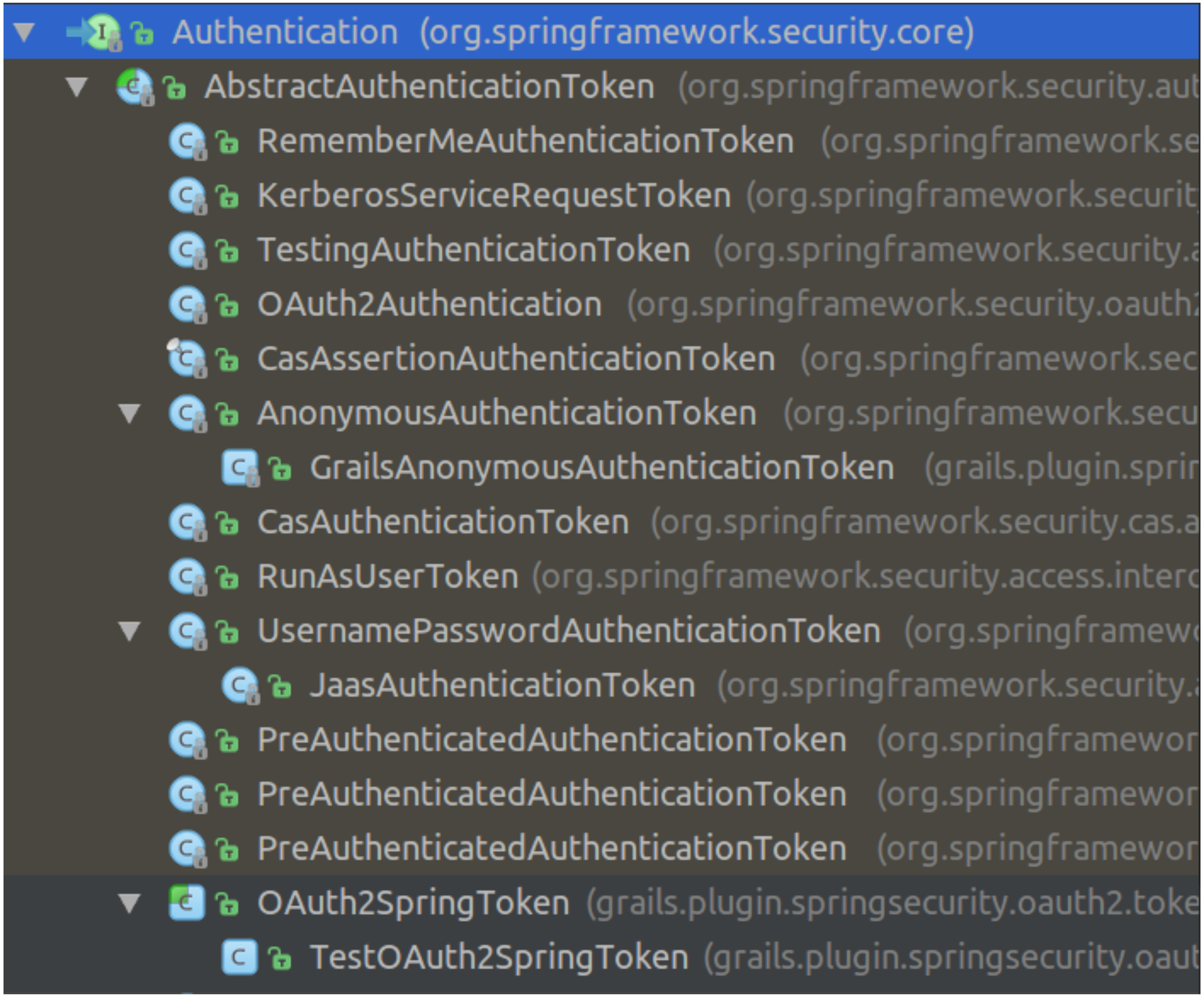
```
public interface AuthenticationProvider {

        Authentication authenticate(Authentication authenticatio
                        throws AuthenticationException;


        boolean supports(Class<?> authentication);
}
```

AuthenticationProvider

- DaoAuthenticationProvider
- GrailsAnonymousAuthenticationProvider
- RunAsImplAuthenticationProvider
- KerberosServiceAuthenticationProvider
- CasAuthenticationProvider
- FacebookAuthProvider
- ...

Authentication

```java
public interface Authentication extends Principal, Serializable
        Collection<? extends GrantedAuthority> getAuthorities();
        Object getCredentials();
        Object getDetails();
        Object getPrincipal();
        boolean isAuthenticated();
        void setAuthenticated(boolean isAuthenticated)
                                    throws IllegalArgumentException;
}
```



```
▼ Authentication (org.springframework.security.core)
    ▼ AbstractAuthenticationToken (org.springframework.security.aut
        RememberMeAuthenticationToken (org.springframework.se
        KerberosServiceRequestToken (org.springframework.securit
        TestingAuthenticationToken (org.springframework.security.a
        OAuth2Authentication (org.springframework.security.oauth
        CasAssertionAuthenticationToken (org.springframework.sec
      ▼ AnonymousAuthenticationToken (org.springframework.secu
            GrailsAnonymousAuthenticationToken (grails.plugin.sprin
        CasAuthenticationToken (org.springframework.security.cas.a
        RunAsUserToken (org.springframework.security.access.interc
      ▼ UsernamePasswordAuthenticationToken (org.springframew
            JaasAuthenticationToken (org.springframework.security.a
        PreAuthenticatedAuthenticationToken (org.springframewor
        PreAuthenticatedAuthenticationToken (org.springframewor
        PreAuthenticatedAuthenticationToken (org.springframewor
      ▼ OAuth2SpringToken (grails.plugin.springsecurity.oauth2.toke
            TestOAuth2SpringToken (grails.plugin.springsecurity.oaut
```

Authentication Filter

```
public abstract class AbstractAuthenticationProcessingFilter
     extends GenericFilterBean
        implements ApplicationEventPublisherAware,
            MessageSourceAware {
```

AuthenticationManager

```
public interface AuthenticationManager {
        Authentication authenticate(Authentication authenticatio
                        throws AuthenticationException;
}
```

Spring Security Rest

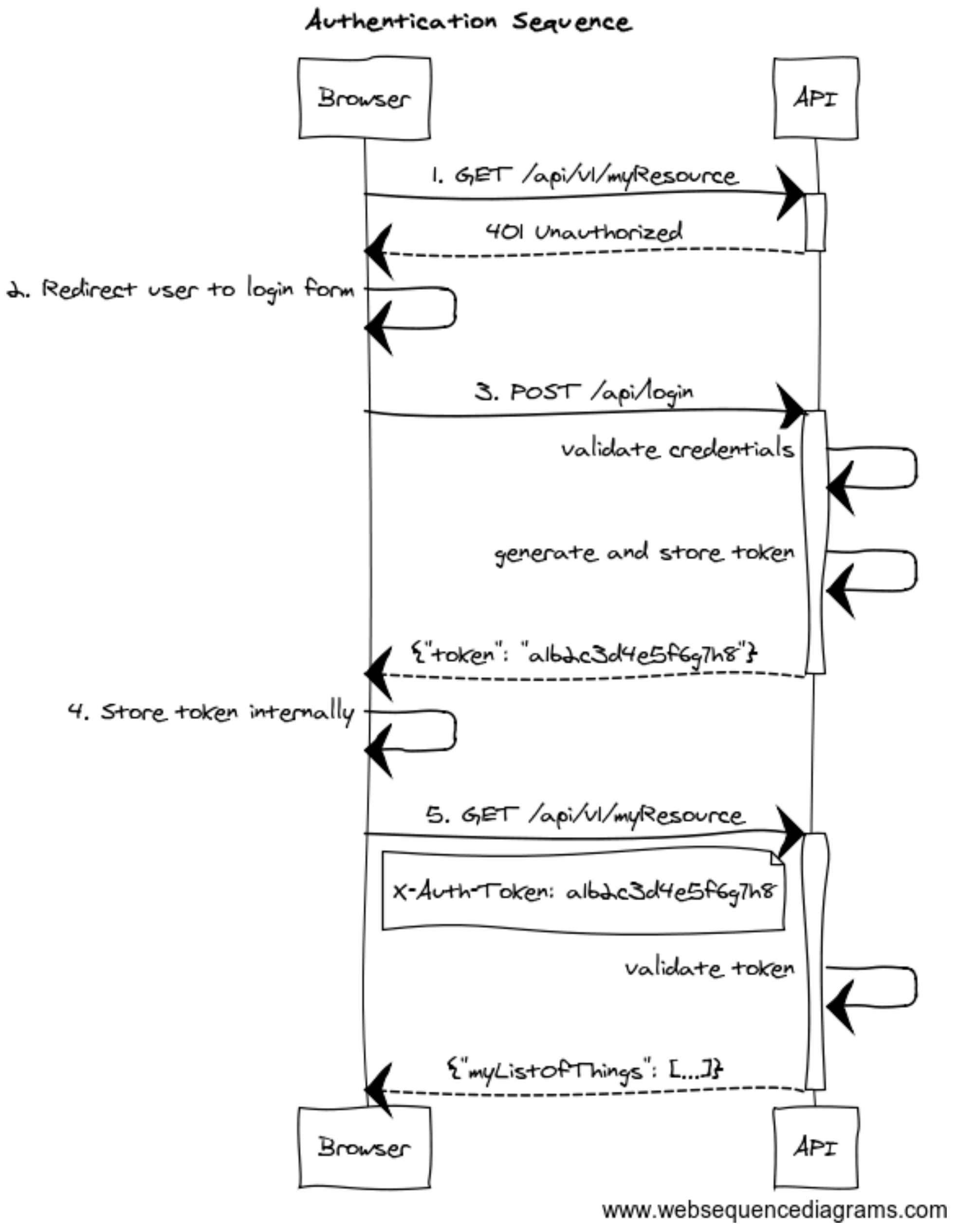
Restful Auth

Token Based

Stateful/Stateless

JWT

RFC6750


Authentication Sequence
Browser
API
1. GET /api/v1/myResource
401 Unauthorized
2. Redirect user to login form
3. POST /api/login
validate credentials
generate and store token
{"token": "a1b2c3d4e5f6g7h8"}
4. Store token internally
5. GET /api/v1/myResource
X-Auth-Token: a1b2c3d4e5f6g7h8
validate token
{"myListofThings": [...]}
Browser
API
www.websequencediagrams.com

Stateful

- memcached
- GORM
- Redis
- Grails Cache

Stateless

JWT Signatures

Encryption

Authentication Provider

- Internal Auth
- Delegation
  - Google
  - Facebook
  - Twitter
  - CAS

Facebook

spring-security-facebook

Grails plugin for Facebook Authentication,
extension to Grails Spring Security Core plugin

- https://github.com/splix/grails-spring-security-facebook
- http://splix.github.io/grails-spring-security-facebook/

Approaches

- Server-Side authentication (FacebookAuthRedirectFilter)
- Transparent cookie based authorization (FacebookAuthCookieTransparentFilter)
- Manual cookie based authentication (FacebookAuthCookieDirectFilter)
- JSON or Android/iOS/desktop authorization (FacebookAuthJsonFilter)

ACL

- http://grails-plugins.github.io/grails-spring-security-acl/
- https://github.com/grails-plugins/grails-spring-security-acl

Access Control List

The **ACL** plugin adds **Domain Object Security** support to a Grails application that uses Spring Security. It depends on the Spring Security Core plugin.

The core plugin and other extension plugins support restricting access to URLs via rules that include checking a user’s authentication status, roles, etc. and the ACL plugin extends this by adding support for restricting access to individual domain class instances. The access can be very **fine-grained** and can define which actions can be taken on an object - these typically include Read, Create, Write, Delete, and Administer but you’re free to define whatever actions you like.

ACLs

- GORM
- Customizable
- Services
- Taglib

```
@PreAuthorize

@PreFilter

@PostAuthorize

@PostFilter
```

```
class ReportService {

   @PreAuthorize("hasPermission(#id, 'com.yourapp.Report', read) or " +
                 "hasPermission(#id, 'com.yourapp.Report', admin)")
   Report getReport(long id) {
      Report.get(id)
   }

   @PreAuthorize("hasRole('ROLE_USER')")
   @PostFilter("hasPermission(filterObject, read) or " +
               "hasPermission(filterObject, admin)")
   List getAllReports(params = [:]) {
      Report.list(params)
   }

   @PreAuthorize("hasPermission(#report, write) or " +
                 "hasPermission(#report, admin)")
   Report updateReport(Report report, params) {
      report.properties = params
      report.save()
      report
   }

   @PreAuthorize("hasPermission(#report, delete) or " +
                 "hasPermission(#report, admin)")
   void deleteReport(Report report) {
      report.delete()
   }
```

```
class AclUtilService {
    void addPermission(Class<?> domainClass, Serializable id,
                recipient, Permission permission) { ... }
    void addPermission(domainObject, recipient, Permission permission) {
    void addPermission(ObjectIdentity oid, recipient,
                Permission permission) { ... }
    void changeOwner(domainObject, String newUsername) { ... }
    void deletePermission(domainObject, recipient, Permission permission)
    void deletePermission(Class<?> domainClass, long id,
                recipient, Permission permission) { ... }
    boolean hasPermission(Authentication authentication,
                domainObject, Permission... permissions) { ... }
    boolean hasPermission(Authentication authentication,
                domainObject, List<Permission> permissions) { ... }
    Acl readAcl(domainObject) { ... }
    Acl readAcl(Class<?> domainClass, id) { ... }
    void deleteAcl(domainObject) { ... }
    protected Sid createSid(recipient) { ... }
}
```

Spring Security Shiro

Spring Security Shiro

The Spring Security **Shiro** plugin adds some support for using a hybrid approach combining Spring Security and Shiro. It currently only supports Shiro **ACLs**, since Spring Security ACLs are very powerful but can be very cumbersome to use, and the Shiro approach is straightforward and simple.

The majority of the authentication and authorization work is still done by Spring Security. This plugin listens for Spring Security authentication events and uses the Spring Security Authentication instance to build and register a ShiroSubject instance. It also removes the Shiro credentials when you explicitly logout.

Permissions

```
grails.plugin.springsecurity.shiro.permissionDomainClassName =
      'com.mycompany.myapp.Permission'
```

```
class Permission {

   User user
   String permission

   static constraints = {
      permission unique: 'user'
   }
}
```

```
import com.mycompany.myapp.MyShiroPermissionResolver

beans = {
   shiroPermissionResolver(MyShiroPermissionResolver)
}
```

```
import org.apache.shiro.SecurityUtils
import org.apache.shiro.subject.Subject

...

Subject subject = SecurityUtils.getSubject()

subject.checkPermission('printer:print:lp7200')

subject.isPermitted('printer:print:lp7200')

subject.checkRole('ROLE_ADMIN')

subject.hasRole('ROLE_ADMIN')

subject.isAuthenticated()

... etc
```

grails-security-bridge

Grails Security Bridge

The Grails Security Bridge plugin is used for providing a decoupled, cross-plugin security interface. This allows you to keep the majority of authentication logic in one plugin, while other plugins can reference a public API interface to retrieve the information needed.

- https://bintray.com/bertramlabs/grails3-plugins/security-bridge
- https://github.com/bertramdev/grails-security-bridge

grails-security-bridge

```
@Secure Annotation
...
interface SecurityBridge {
        def getCurrentUser()
        def getUserIdentity()
        def getCurrentAccount()
        def getAccountIdentity()
        def getCurrentUserDisplayName()
        boolean isLoggedIn()
        boolean isAuthorized(object, action)
        boolean hasRole(role)
        boolean hasPermission(permission, opts)
        boolean hasPermission(permission)
        def storeLocation(request)
        def withUser(identity, Closure code)
        Map createLink(String action)
}
```

Oauth2

(set of plugins)

spring-security-oauth2

- authenticate via oauth2
- multiple providers
  - spring-security-oauth2-google
  - spring-security-oauth2-facebook

```
application.yml

grails:
    plugin:
        springsecurity:
            oauth2:
                active: true
                registration:
                    askToLinkOrCreateAccountUri: '/oauth2/ask'
                    roleNames: ['ROLE_USER']
```

- **askToLinkOrCreateAccountUri:** The URI that is called to aks the user to either create a new account or link to an existing account

- **roleNames:** A list of role names that should be automatically granted to an OAuth User. The roles will be created if they do not exist

```
class MyGrailsUser {
     ...
     static hasMany = [oAuthIDs: OAuthID]
}
```

Add the oAuthIDs to your user object

oauth2-provider

The OAuth2 plugin adds OAuth 2.0 support to a Grails application that uses Spring Security.

It depends on Spring Security Core plugin.

Authentication Providers

- LDAP
- CAS
- Kerberos

Tools

- appinfo
- ui

Enhancement

- acl
- jaxrs

Token/Stateless

- rest

oauth2

- oauth2-provider
- oath2
  - oauth2-facebook
  - oauth2-google

Adapter

- shiro
- security-bridge

Resources

- grails.org
- User Guide
  (RTFM: Read The Fantastic Manual)
- Stack Overflow
- Slack
- Github!
- twitter

Connect with Us

grails.org
slack-signup.grails.org
grailsblog.ociweb.com
ociweb.com/grails
@grailsframework
@objectcomputing

OCI | HOME TO GRAILS

Thank you for attending!

Please visit our table for giveaways and to meet the team!

ociweb.com/grails
info@ociweb.com

OCI | HOME TO GRAILS