Internet Applications Design and Implementation

2015 - 2016 - 1st edition

(Lecture 10 - Security of Internet Applications)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Security of Internet Applications

- Layers of internet application security
  - Network Level (network security and system identification)
  - System Level (firewalls, VPNs, SSL, DMZ)
  - Application Level
    - Authentication
    - Access control
    - Information flow

Security Attacks

Home
About OWASP
Acknowledgements
Advertising
AppSec Events
Books
Brand Resources
Chapters
Donate to OWASP
Downloads
Funding
Governance
Initiatives
Mailing Lists
Membership
Merchandise
News
Community portal
Presentations
Press

Category | Discussion

Category:Attack

This category is for tagging common types of application security attacks.

What is an attack?

Attacks are the techniques that attackers use to exploit the vulnerabilities in application.

All attack articles should follow the Attack template.

Examples:

- Brute Force: Is an exhaustive attack that works by testing every possible
- Cache Poisoning: Is an attack that seeks to introduce false or malicious
- DNS Poisoning: Is an attack that seeks to introduce false DNS address i

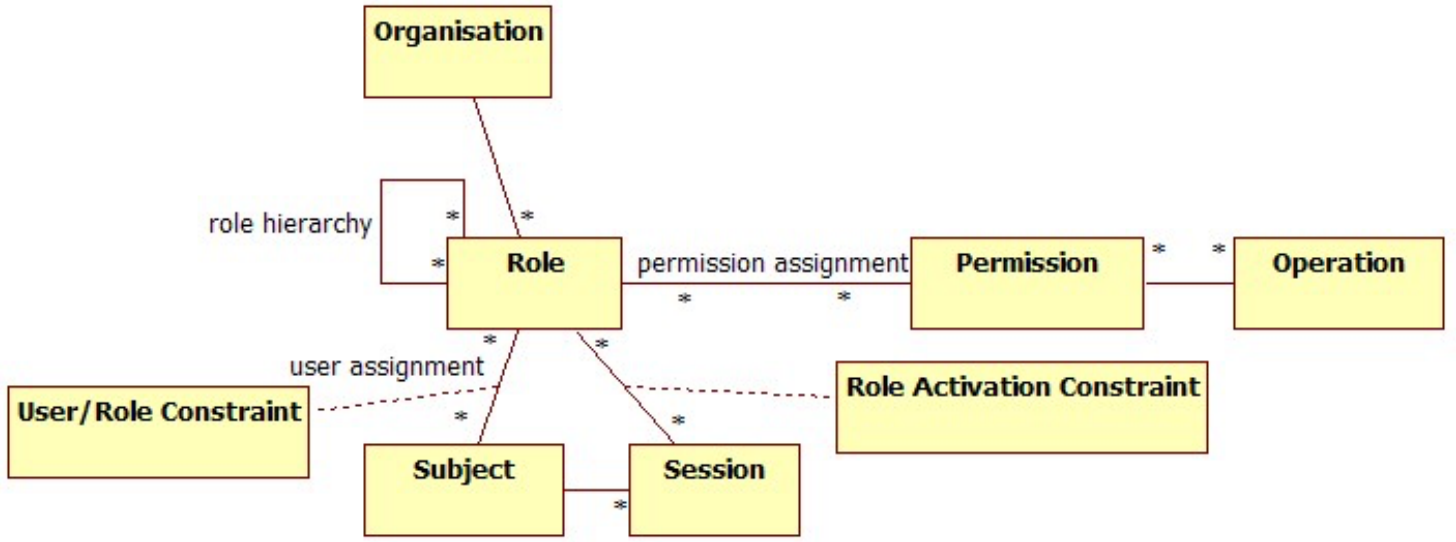
Note: many of the items marked vulnerabilities from CLASP and other place

- Log injection
- Resource exhaustion
- Reflection injection
- Reflection attack in an auth protocol

https://www.owasp.org/index.php/Category:Attack

Application Level - Role Base Access Control

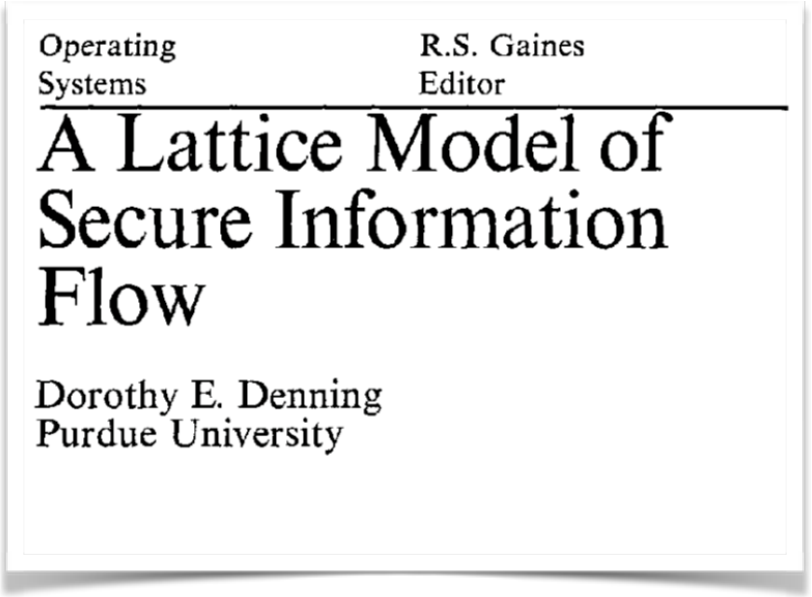
- Standard approach to authorise users to perform operations in software systems.
  - Roles are specified for system related tasks.
  - Permissions are assigned to specific roles.
  - Users are assigned rules according to their profile/function.



https://upload.wikimedia.org/wikipedia/en/c/c3/RBAC.jpg

Application Level - Role Base Access Control

- Standard approach to authorise users to perform operations in software systems.
  - Roles are specified for system related tasks.
  - Permissions are assigned to specific roles.
  - Users are assigned rules according to their profile/function.
- Roles can be combined in hierarchies or lattices
- RBAC provides a framework for a verification based on interceptors and filters
- Roles can also be parameterised and depend on actual data.


Operating Systems
R.S. Gaines Editor

A Lattice Model of Secure Information Flow

Dorothy E. Denning
Purdue University

Security of Internet Applications (Vocabulary)

- Principal - Who is the entity behind a particular request

```
@RequestMapping("/messages/inbox")
public ModelAndView findMessagesForUser(@AuthenticationPrincipal CustomUser user) {

    // .. find messages for this user and return them ...
}
```

- Authentication: Certify that a given set of credentials identify one principal

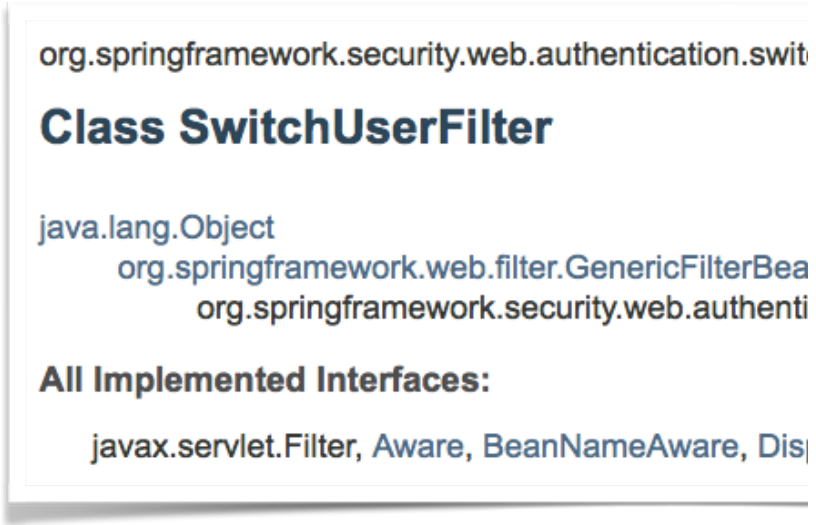
```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth
        .jdbcAuthentication()
            .dataSource(dataSource)
            .withDefaultSchema()
            .withUser("user").password("password").roles("USER").and()
            .withUser("admin").password("password").roles("USER", "ADMIN");
}
```

Security of Internet Applications (Vocabulary)

- Authorisation - Check if a given principal has rights to access a given piece of information

```
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .antMatchers("/resources/**", "/signup", "/about").permitAll()
            .antMatchers("/admin/**").hasRole("ADMIN")
            .antMatchers("/db/**").access("hasRole('ROLE_ADMIN') and hasRole('ROLE_DBA')")
            .anyRequest().authenticated()
            .and()
        // ...
        .formLogin();
}
```

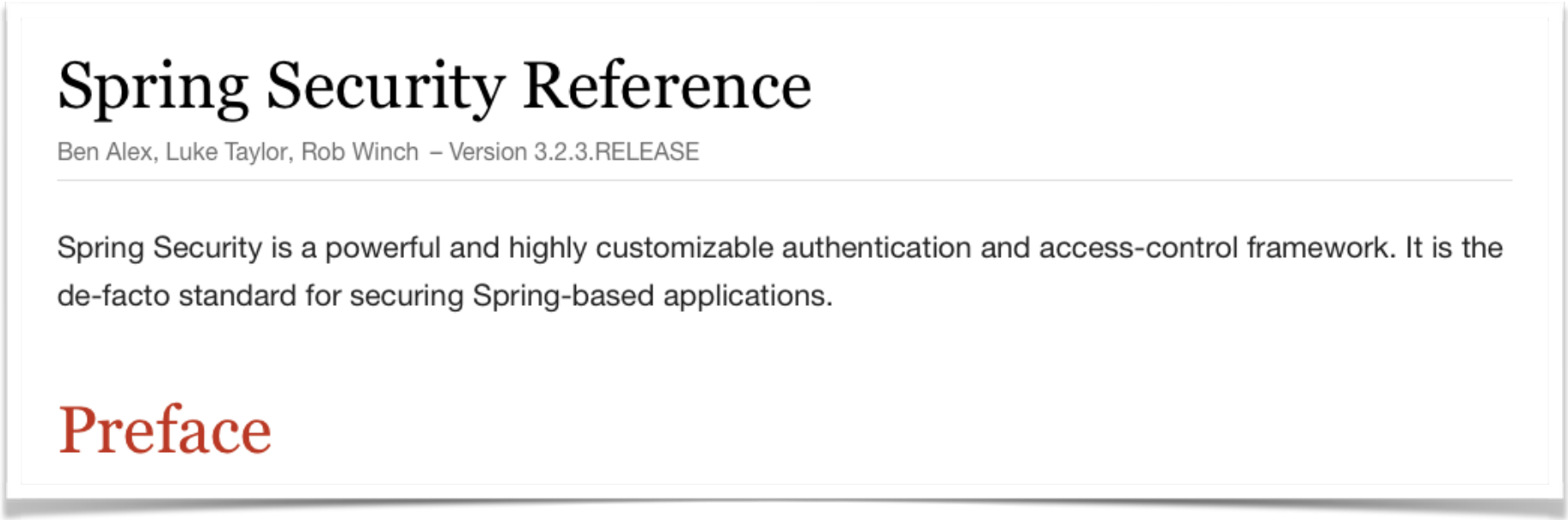
- Delegation - to be able to temporarily assign one principals capabilities' to another principal (e.g. sudo)



http://docs.spring.io/autorepo/docs/spring-security/3.2.1.RELEASE/apidocs/org/springframework/security/web/authentication/switchuser/SwitchUserFilter.html

Using frameworks for security

- Frameworks such as Spring Security promote the reuse of (correct) code, good practices, and great number of base features.



Spring Security Reference

Ben Alex, Luke Taylor, Rob Winch – Version 3.2.3.RELEASE

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

Preface

http://docs.spring.io/spring-security/site/docs/3.2.8.RELEASE/reference/htmlsingle/

Using frameworks for security

- Frameworks such as Spring Security promote the reuse of (correct) code, good practices, and great number of base features.
  - HTTP BASIC authentication headers (an IETF RFC-based standard)
  - HTTP Digest authentication headers (an IETF RFC-based standard)
  - HTTP X.509 client certificate exchange (an IETF RFC-based standard)
  - LDAP (a very common approach to cross-platform authentication needs, especially in large environments)
  - Form-based authentication (for simple user interface needs)
  - OpenID authentication
  - Authentication based on pre-established request headers (such as Computer Associates Siteminder)
  - JA-SIG Central Authentication Service (otherwise known as CAS, which is a popular open source single sign-on system)
  - Transparent authentication context propagation for Remote Method Invocation (RMI) and HttpInvoker (a Spring remoting protocol)
  - Automatic "remember-me" authentication (so you can tick a box to avoid re-authentication for a predetermined period of time)
  - Anonymous authentication (allowing every unauthenticated call to automatically assume a particular security identity)
  - Run-as authentication (which is useful if one call should proceed with a different security identity)
  - Java Authentication and Authorization Service (JAAS)
  - JEE container autentication (so you can still use Container Managed Authentication if desired)
  - Kerberos

Authentication - Simple starter code

- The default security setting provides that all requests are protected, a login form is created, RESTful interface for login/logout…

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .inMemoryAuthentication()
                .withUser("user").password("password").roles("USER");
    }
}
```

Authentication - Simple starter code

- The default security setting provides that all requests are protected, a login form is created, RESTful interface for login/logout…

```
@Autowired
private DataSource dataSource;

@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth
        .jdbcAuthentication()
            .dataSource(dataSource)
            .withDefaultSchema()
            .withUser("user").password("password").roles("USER").and()
            .withUser("admin").password("password").roles("USER", "ADMIN");
}
```

Authentication - Simple starter code

- The default security setting provides that all requests are protected, a login form is created, RESTful interface for login/logout…

```
@Override
protected void configure(HttpSecurity http) {
        http
            .authorizeRequests()
            .antMatchers("/**").hasRole("USER")
            .and()
            .openidLogin()
            .permitAll();
}

@Override
protected void configure(AuthenticationManagerBuilder auth)(
               AuthenticationManagerBuilder auth) throws Exception {
        auth
        .inMemoryAuthentication()
.withUser("https://www.google.com/accounts/o8/id?id=lmkCn9xzPdsxVwG7pjYMuDgNNdASFmobNkcRPaWU")
        .password("password")
        .roles("USER");
}
```

http://docs.spring.io/autorepo/docs/spring-security/3.2.6.RELEASE/apidocs/org/springframework/security/config/annotation/web/configurers/openid/OpenIDLoginConfigurer.html

Basic Spring Security Guaranties

There really isn't much to this configuration, but it does a lot. You can find a summary of the features below:

- Require authentication to every URL in your application
- Generate a login form for you
- Allow the user with the **Username** *user* and the **Password** *password* to authenticate with form based authentication
- Allow the user to logout
- CSRF attack prevention
- Session Fixation protection
- Security Header integration
  - HTTP Strict Transport Security for secure requests
  - X-Content-Type-Options integration
  - Cache Control (can be overridden later by your application to allow caching of your static resources)
  - X-XSS-Protection integration
  - X-Frame-Options integration to help prevent Clickjacking
- Integrate with the following Servlet API methods
  - HttpServletRequest#getRemoteUser()
  - HttpServletRequest.html#getUserPrincipal()
  - HttpServletRequest.html#isUserInRole(java.lang.String)
  - HttpServletRequest.html#login(java.lang.String, java.lang.String)
  - HttpServletRequest.html#logout()

CSRF - token issued by the server / recognised by the same server

6. Cross Site Request Forgery (CSRF)

This section discusses Spring Security's Cross Site Request Forgery (CSRF) support.

6.1. CSRF Attacks

Before we discuss how Spring Security can protect applications from CSRF attacks, we will explain what a CSRF attack is. Let's take a look at a concrete example to get a better understanding.

Assume that your bank's website provides a form that allows transferring money from the currently logged in user to another bank account. For example, the HTTP request might look like:

```
POST /transfer HTTP/1.1
Host: bank.example.com
Cookie: JSESSIONID=randomid; Domain=bank.example.com; Secure; HttpOnly
Content-Type: application/x-www-form-urlencoded

amount=100.00&routingNumber=1234&account=9876
```

CSRF - token issued by the server / recognised by the same server

- Template example…

```
<div th:if="${#httpServletRequest.remoteUser}!=null">
  <label>User: </label><span th:text="${#httpServletRequest.remoteUser}"></span>
  <form style="display:inline-block" th:action="@{/logout}" method="post">
      <input type="submit" value="Sign Out"/>
  </form>
</div>

<div th:if="${#httpServletRequest.remoteUser} == null" >
  <form th:action="@{/}" method="post">
      <label> User : <input type="text" name="username"/> </label>
      <label> Password: <input type="password" name="password"/> </label>
      <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
      <input type="submit" value="Sign In"/>
  </form>

</div>
```

CSRF - token issued by the server / recognised by the same server

- Actual page example

```
<div class="container">
  <div>
    <form method="post" action="/">
      <label>
        " User : "
        <input type="text" name="username">
      </label>
      <label>
        " Password: "
        <input type="password" name="password">
      </label>
      <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}">
      <input type="submit" value="Sign In">
      <input type="hidden" name="_csrf" value="f067a25f-c6f5-4de0-b6ad-ea7416986b22">
    </form>
  </div>
</div>
```

Authorization - Access control

- Ensures that any request to the application requires the user to be authenticated
- Allows users to authenticate with form based login
- Allows users to authenticate with HTTP Basic authentication

```
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .anyRequest().authenticated()
            .and()
        .formLogin()
            .and()
        .httpBasic();
}
```

Authorization - Access control

- Refine the access conditions for each request

```
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .antMatchers("/resources/**", "/signup", "/about").permitAll()
            .antMatchers("/admin/**").hasRole("ADMIN")
            .antMatchers("/db/**").access("hasRole('ROLE_ADMIN') and hasRole('ROLE_DBA')")
            .anyRequest().authenticated()
            .and()
        // ...
        .formLogin();
}
```

Authorization - Access control

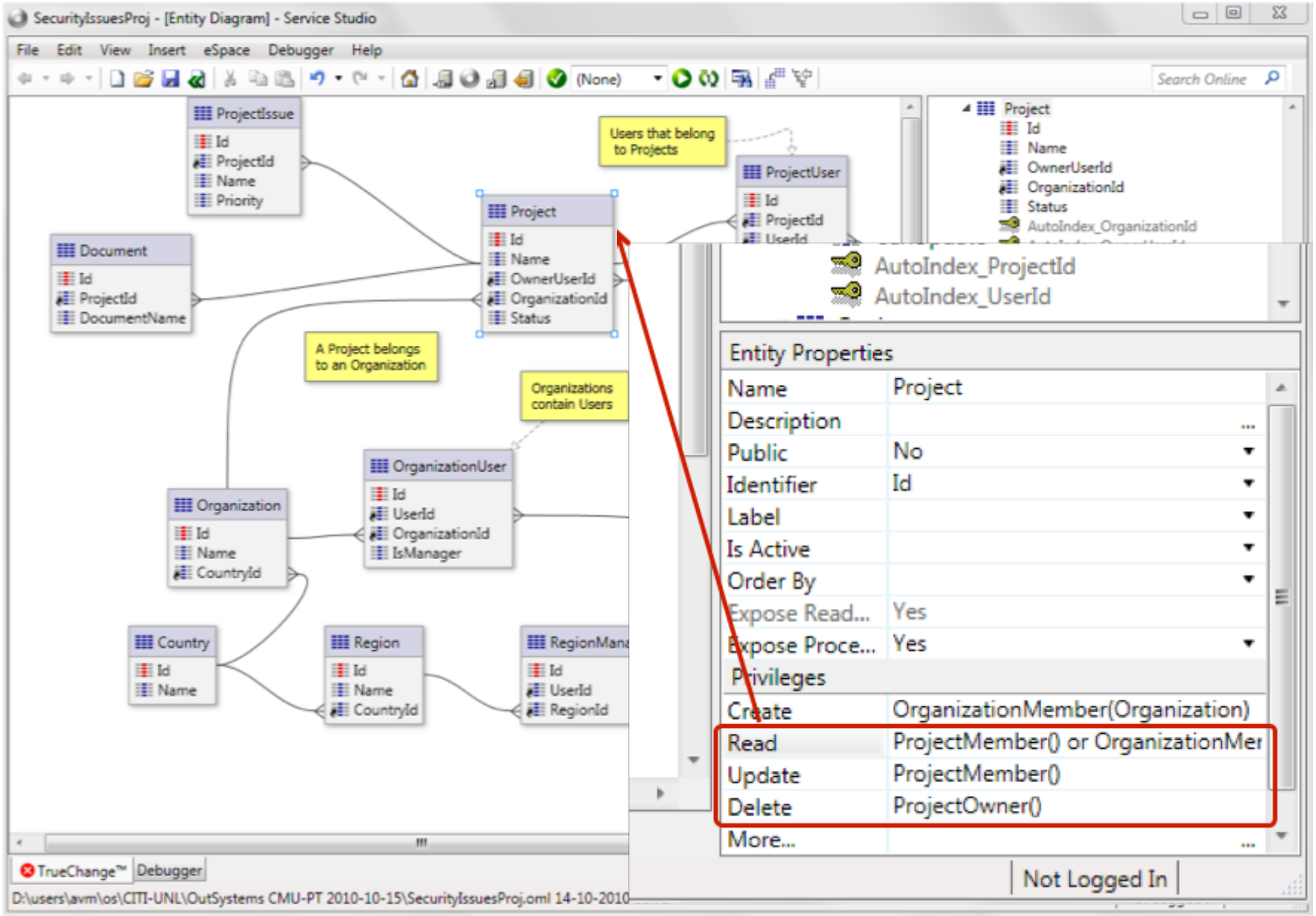
- And configure the behaviour and elements of the framework itself.

```
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .antMatchers("/").permitAll()
            .anyRequest().authenticated()
            .and()
        .formLogin()
            .loginPage("/")
            .defaultSuccessUrl("/")
            .permitAll()
            .and()
        .logout()
            .logoutSuccessUrl("/")
            .permitAll();
}
```

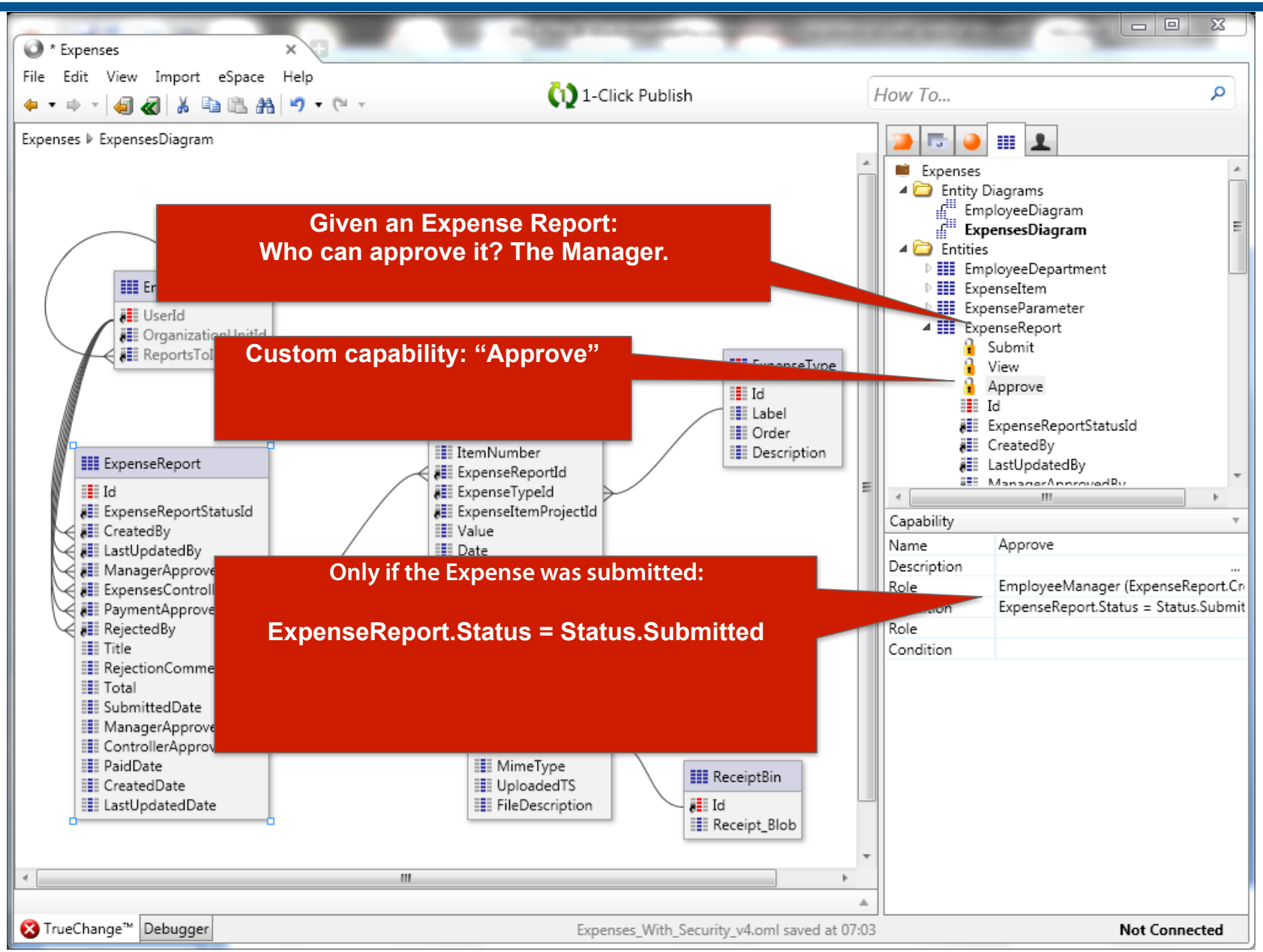
Model Based Access Control

- Security conditions are many times hidden in query filters and program conditions.
- Developer defined roles that depend on the state of the application. (e.g. ModeratorOf(…)).
- Authorisations that depend on the state of the target entity (status == submitted)
- Capabilities defined by the developer (not hardwired to the programming elements)

Model Based Access Control



Model Based Access Control - Custom Cap.



Model Based Access Control - Spring?

```
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .anyRequest().authenticated()
            .and()
            .antMatchers(“/admin/hotels/{id}”)
            .access(id => managerOfHotel(id))
            .access(id => hotels.findOne(id).isApproved())
//…
}
```