JEE Security Structure Part 1

Ari Ayvazyan

29.09.2014

Contents

1 JEE Security Structure Part 1 **2**

1.1 Introduction to Security Architecture 2
1.2 Authentication 3
1.3 Authorization 3
1.4 Deployment Descriptors 4
1.4.1 web.xml 4
1.4.2 (vendor-specific).xml 7
1.5 Principals 8
1.6 Credential 8
1.7 Groups 8
1.8 Roles 9
1.9 Realms 9
1.10 Implementation Sample 10
1.10.1 Objective 10
1.10.2 setup 11
1.11 Output Escaping 13
1.12 Frameworks 13
1.12.1 Shiro 13
1.12.2 Spring 13
1.12.3 JAAS - Java Authentication and Authorization Service 13
1.13 Whats to come in Part 2 (Adrian) 14

Chapter 1

JEE Security Structure Part 1

1.1 Introduction to Security Architecture

Most web applications have a few things in common:
They need to figure out who is the user that is using the application and what is he allowed to do and see.

A typical application has more than one security layer, it may be protected by only being available from a specified network or VPN. In addition there usually is some kind of identity determination followed by a SQL user with permission to query only the required functions and data sets.
On top of this, there should be output escaping to ensure that a attacker, who is able to manipulate the output for users, is limited in the harm he is able to cause.

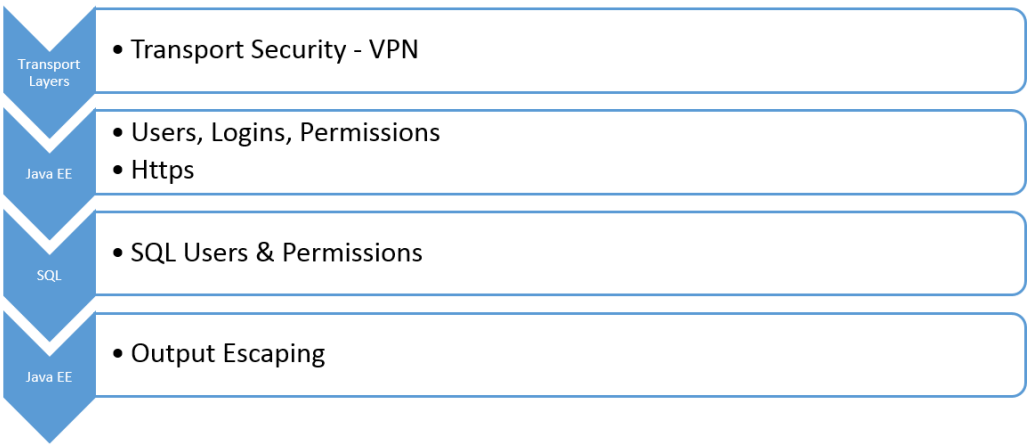

Figure 1.1: Security Layers in a common JEE application

According to Oracle[3], there are two ways to implement such access control functionality with Java EE:

1. Programmatic
2. Declarative (this includes Annotations and XML-Files)

While the programmatic implementation offers a wider range of customization, the declarative provides a well structured and easy to use approach.

1.2 Authentication

Authentication describes the identification process. This is mostly done by asking for a user-name & password or sending a Token/Hash.

1.3 Authorization

Authorization is what happens after you are authenticated.
It deals with the question of what a authenticated person is allowed to do.
Authorization may be applied to URLs or resources like Beans and Servlets.

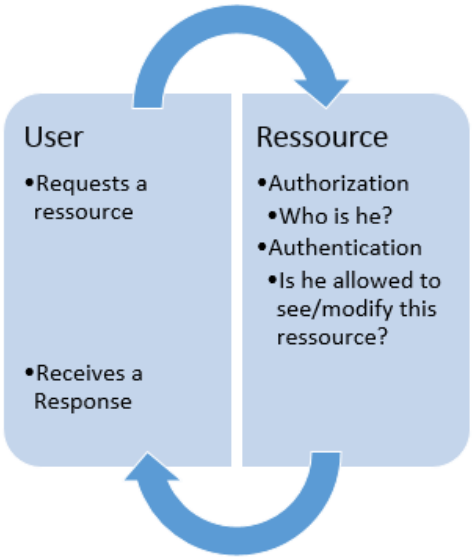

Figure 1.2: Authentication & Authorization

1.4 Deployment Descriptors

A Deployment Descriptor describes how a Java EE application should be deployed.
They contain information about security constraints, accessibility and resource references.

Deployment Descriptors are XML-Files that are by default located in the /WEB-INF/ directory.
The following deployment descriptors may be found here:

- web.xml
- <vendor-specific>.xml (E.g. when using Glassfish: glassfish-web.xml)

1.4.1 web.xml

The Web.xml file stores apart from usual deployment information like servlet mappings also security related information about:

- Protected Resources
- Security Roles
- Authentication methods

The following XML snippets are located within the <web-app></web-app> tag.

Protected Resources

It is possible to limit access to resources by defining a security constraint on the URL or by securing the resource itself.
E.g. to protect the /primes/ URL with all its subdirectories we would have to use the following code:

Securing a URL

```
<security-constraint>
        <web-resource-collection>
                <web-resource-name>primes
                </web-resource-name>
<!--Include /primes/ including all following subfolders-->
                <url-pattern>/primes/*</url-pattern>
<!-- This would result in a security leak because
        there are more http-methods than GET and POST-->
<!-- by defining no http-method at all,
        everything will be blocked-->
                <http-method>GET</http-method>
                <http-method>POST</http-method>
        </web-resource-collection>

        <auth-constraint>
                <role-name>view_role</role-name>
        </auth-constraint>
</security-constraint>
```

In result, only a user with the role view_role is allowed to access the defined resources.

Security Roles

A Security Role was used in the last subsection "Protected Resources", we declared that only users with the role "view_role" are allowed to view the restricted URL.
A Security Role is a abstract layer in front of the container, it defines a identifier which we can use for constraints. This identifier is then used by the container to specify its meaning by telling who is part of this Security Role.

Defining a Security Role

```
<security-role>
        <description>This role has view access</description>
        <role-name>view_role</role-name>
</security-role>
```

Authentication Methods

There are several kinds of Authentication Methods with different security behaviors.
BASIC Authentication opens a login prompt when a user tries to access the secured URL, it is simple to implement but insecure. BASIC Authentication sends the user's credentials unencrypted to the server.

Defining BASIC Authentication

```
<login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>Java EE Login</realm-name>
</login-config>
```

1.4.2 (vendor-specific).xml

Most containers use in addition to the web.xml a vendor specific XML file that is usually located in the same directory with web.xml.
E.g. Glassfish calls this file "glassfish-web.xml" while Tomcat has named it "context.xml".

The following settings can be configured there:

- Principal – Role Mapping
- Group – Role Mapping
- Other container specific configuration.

User Role Mapping

It is possible to Map Users (Principals) to Roles. This setting is container dependent.

Principal mapping with Glassfish

```
<glassfish-web-app>
<security-role-mapping>
        <role-name>view_role</role-name>
        <principal-name>Blitzmann</principal-name>
</security-role-mapping>
</glassfish-web-app>
```

This setting maps the user "Blitzmann" to the role "view_role".

Group Role Mapping

It is possible to Map Users (Principals) to Roles. This setting is container dependent.

Principal mapping with Glassfish

```
<glassfish-web-app>
<security-role-mapping>
<role-name>view_role</role-name>
<group-name>view_group</group-name>
</security-role-mapping>
</glassfish-web-app>
```

This setting maps the group "view_group" to the role "view_role".

1.5 Principals

A Principal is a identity that can be authenticated.
E.g. a Unique user name
Principals are handled by the container and need to be configured there. This includes storage and queries

1.6 Credential

A Credential is defined as information that is used to authenticate a Principal.
E.g. a Password
Credentials are just like Principals handled by the container.

1.7 Groups

Groups and Principals can be mapped together. This information is handled by the container and by that not accessible by the application. To make this information available to the server, Groups can be mapped to Roles.
Group-Role mappings are usually defined in the (vendor-specific).xml file.

1.8 Roles

A Role is a abstract layer in front of the container that can be accessed application wide, they are declared in the web.xml file
Roles can be mapped to Principals and Groups via the (vendor-specific).xml file.

1.9 Realms

Realms are also known as Security policy domains, they store and provide information about Principals, their Credentials and their Groups.
A Realm may take use of a database, a file structure, a connection to a service...

Realms are handled by the container, some come with pre-configured realms.

1.10 Implementation Sample

The following sample may be found at this URL:
https://github.com/aayvazyan-tgm/JavaEESecurityExample

1.10.1 Objective

The objective of our sample is to deny access to our service, that is reachable at .../primes/, to everybody except users that are granted permission by possessing the "view_role" role.

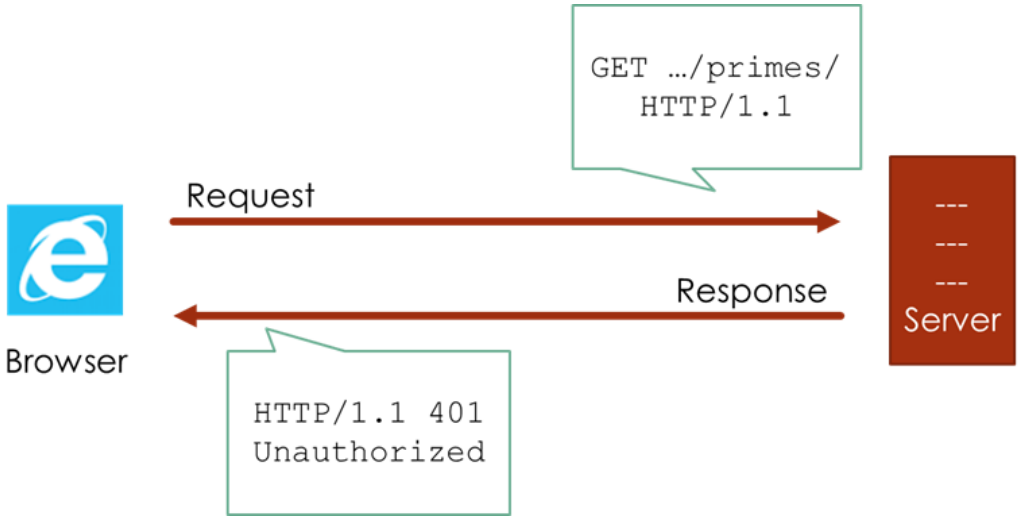

Figure 1.3: The user tries to access a resource without authentication

To figure out which user possesses the "view_role" role, we first have to authenticate the user. This shall happen with a login promt. After transmitting the credentials to the server and having the correct role, the user should be able to access the resource.

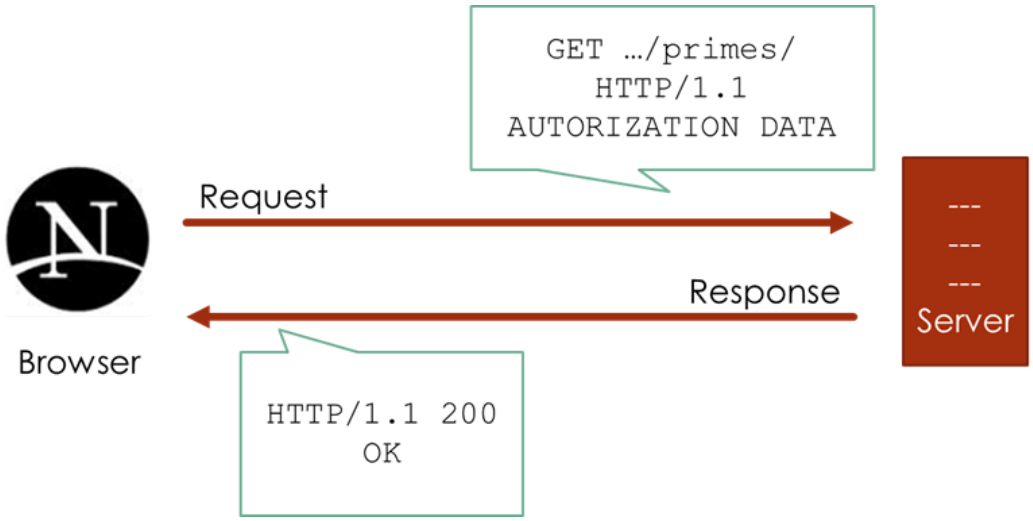

Figure 1.4: The user sends authentication data with his request

1.10.2 setup

A Glassfish server is used as the container.
The user "Blitzmann" was created by navigating to the GlassFish Server Administration Console, then to Configuration -> server-config -> Security -> Realms -> file -> add Property and entering "Blitzmann" as Name and the desired password as Value.
The rest is configured in the web.xml and glassfish-web.xml

web.xml

```
<security-constraint>
<web-resource-collection>
        <web-resource-name>primes</web-resource-name>
        <!--Include all urls that are starting with /primes/ -->
        <url-pattern>/primes/*</url-pattern>
        <!-- This would result in a security leak because
                there are more http-methods than get and post-->
        <!-- by defining no http-method at all
                everything will be blocked-->
        <!--http-method>GET</http-method>
        <http-method>POST</http-method-->
</web-resource-collection>
        <auth-constraint>
                <role-name>view_role</role-name>
        </auth-constraint>
</security-constraint>

<login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>file</realm-name>
</login-config>

<security-role>
    <description>This role has view access</description>
    <role-name>view_role</role-name>
</security-role>
```

glassfish-web.xml

```
<glassfish-web-app>
<security-role-mapping>
<role-name>view_role</role-name>
<group-name>view_group</group-name>
<principal-name>Blitzmann</principal-name>
</security-role-mapping>
</glassfish-web-app>
```

1.11 Output Escaping

Security vulnerabilities happen to be found in the best applications, it is important to have more layers of security that prevent attackers from reading or modifying sensitive information.
E.g. we have a guestbook, everybody is allowed to write a entry. To prevent a attacker from writing custom HTML into our site, we may use input escaping. But if the attacker has managed to someway access our database in a different way, where no input escaping is applied, he would be able to perform his attack on our users.
To prevent him from doing so, we have to check our output (Output Escaping). This means that we only allow data to be passed that makes sense, so we do not allow HTML elements in this case.
It is important to use a framework to do so, they are proven to work, tested and can even save working hours.

1.12 Frameworks

1.12.1 Shiro

Offers: Authentication, Authorization, Cryptography
Simple to use

1.12.2 Spring

Offers: Authentication, Authorization, Cryptography
Very structured

1.12.3 JAAS - Java Authentication and Authorization Service

Offers: Authentication, Authorization, Cryptography
Included in Java SE since Java 1.4 (javax.security.auth)

1.13 Whats to come in Part 2 (Adrian)

- Working with Digital Certificates
- Securing Application Clients
- Security with Enterprise Beans
- Further Framework Information

Bibliography

[1] Java Security: Sicherheitslücken identifizieren und vermeiden,
Marc Schönefeld, 1. edition 2011
Publisher: Hüthig Jehle Rehm GmbH, Heidelberg.
ISBN/ISSN 978-3-8266-9105-8

[2] Enterprise Java Security: Building Secure J2EE Applications,
Marco Pistoia, Nataraj Nagaratnam, Larry Koved, Anthony Nadalin,
1. edition 2004
Publisher: Addison-Wesley Professional.
ISBN/ISSN: ISBN 0-321-11889-8

[3] Official JavaEE Documentation, Oracle,
29.09.2014 http://docs.oracle.com/javaee/7/tutorial/partsecurity.htm#GIJRP
Java EE 6,
Dirk Weil, 1. edition 2012
Publisher: entwickler.press
ISBN 978-3-86802-077-9

[4] Java EE 6 Cookbook for Securing, Tuning, and Extending Enterprise Applications,
Mick Knutson,
1. edition June 2012
Publisher: Addison-Wesley Professional.
ISBN/ISSN: ISBN 9781849683166

[5] JavaOne 2014: The Anatomy of a Secure Web Application Using Java,
Shawn McKinney & John Field, September 29, 2014
San Francisco