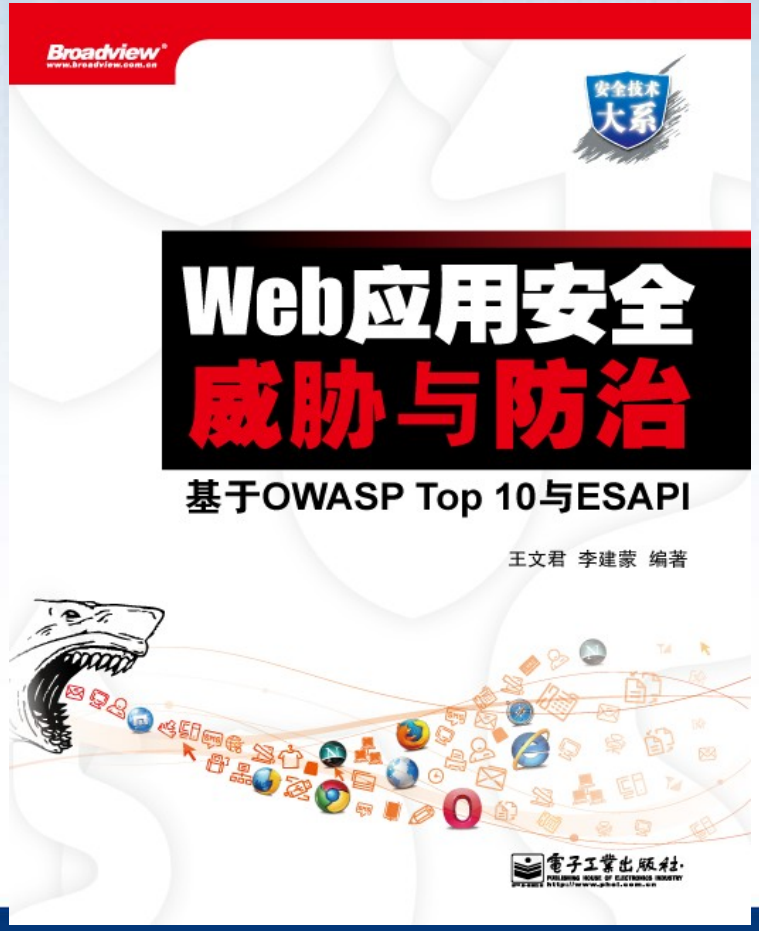
Secure Your RESTful API

Wenjun Wang

Share – Learn - Secure

About me

- HP Software Security Architect & OWASP Shanghai Chapter lead
- Shanda.wang@owasp.org.cn



Agenda

- REST brief introduction
- Secure your RESTful API
- Some attack examples

OWASP
Open Web Application Security Project

Chapter 1
REST brief introduction

OWASP
Open Web Application Security Project

Representational State Transfer

- The URL identifies the resource
- Click on the url (resource) in page (hypermedia)
  - html page is transferred to the browser
  - Representational State transfer occurs

REST Tenets

- Resources (nouns)
  - Identified by a URI, For example:
    - http://www.parts-depot.com/parts
- Methods (verbs) to manipulate the nouns
  - Small fixed set:
    - GET, PUT, POST, DELETE
      - Read, Update, Create, Delete
- **Representation** of the Resource
  - data and state transferred between client and server
  - XML, JSON…
- Use verbs to exchange application state and representation

method

resource

Request: GET
http://localhost:8080/RestfulCustomer/webresources/model.customer/1

Status: 200 (OK)

Time-Stamp: Fri, 14 Dec 2012 02:19:34 GMT

Received:
{"name":"Jumbo Eagle Corp","state":"FL","customerId":1,
"addressline1":"111 E. Las Olivas Blvd","addressline2":"Suite 51",
"city":"Fort Lauderdale","phone":"305-555-0188","fax":"305-555-0189",
"email":"jumboeagle@example.com","creditLimit":100000
}

representation

OWASP
Open Web Application Security Project

Use standard HTTP method

- Example
  - `GET /store/customers/123456`

HTTP Method	Operation Performed
GET	Get a resource (Read a resource)
POST	Create a resource
PUT	Update a resource
DELETE	Delete a resource

OWASP
Open Web Application Security Project

Example

- /orders
  - GET - list all orders
  - POST - submit a new order

  /orders/{order-id}
  - GET - get an order representation
  - PUT - update an order
  - DELETE - cancel an order

  /orders/average-sale
  - GET - calculate average sale
- /customers
  - GET - list all customers
  - POST - create a new customer

  /customers/{cust-id}
  - GET - get a customer representation
  - DELETE- remove a customer

  /customers/{cust-id}/orders
  - GET - get the orders of a customer

Multiple Representations

- Offer data in a variety of formats, for different needs
  - XML /JSON/(X)HTML

- Support **content negotiation**
  - Accept header

```
GET /foo
Accept: application/json
```

  - Response header
  - Content-Type application/xml

OWASP
Open Web Application Security Project

Key benefit for REST

- Server side
  - Uniform Interface
  - Cacheable
  - Scalable
- Client side
  - Easy to experiment in browser
  - Broad programming language support
  - Choice of data formats

OWASP
Open Web Application Security Project

Chapter 2
Secure your RESTful API

OWASP
Open Web Application Security Project

Authentication: Configure web.xml

```
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>admin</realm-name>
</login-config>
```

- **Login-config:**
  - **defines how HTTP requests should be authenticated**
- **Auth-method:**
  - **BASIC, DIGEST, FORM, CLIENT_CERT. corresponds to Basic, Digest, Form and Client Certificate authentication, respectively.**

OWASP
Open Web Application Security Project

Authentication: Configure web.xml

```
<security-constraint>
   <web-resource-collection>
     <url-pattern>/secure/*</url-pattern>
     <http-method>POST</http-method>
  </web-resource-collection>
...
```

- **security constraint**
  - **defines access privileges to a collection of resources**
- **url-pattern:**
  - **URL pattern you want to secure**
- **Http-method:**
  - **Methods to be protected**

OWASP
Open Web Application Security Project

Authentication: Configure web.xml

```
<security-constraint>
...
  <auth-constraint>
     <description>only let admin login </description>
     <role-name>admin</role-name>
  </auth-constraint>
```

- **auth-constraint:**
  - **names the roles authorized to access the URL patterns and HTTP methods declared by this security constraint**

OWASP
Open Web Application Security Project

Encryption: Configure web.xml

```
<security-constraint>
...
  <user-data-constraint>
    <description>SSL</description>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
 </user-data-constraint>
</security-constraint>
```

- **user-data-constraint: NONE, INTEGRAL, or CONFIDENTIAL**
  - **how the data will be transported between client and server**

OWASP
Open Web Application Security Project

Use case 1

HTTP GET operation on a set of web resources should be accessible only by an user with the role "Employee".

```
01. <security-constraint>
02.   <display-name>Restricted GET To Employees</display-name>
03.   <web-resource-collection>
04.     <web-resource-name>Restricted Access - Get Only</web-resource-name>
05.     <url-pattern>/restricted/employee/*</url-pattern>
06.     <http-method>GET</http-method>
07.   </web-resource-collection>
08.   <auth-constraint>
09.     <role-name>Employee</role-name>
10.   </auth-constraint>
11.   <user-data-constraint>
12.     <transport-guarantee>NONE</transport-guarantee>
13.   </user-data-constraint>
14. </security-constraint>
```

OWASP
Open Web Application Security Project

Use case 2

We would like to utilize *FORM* based authentication mechanism.

```
1. <login-config>
2.   <auth-method>FORM</auth-method>
3.   <form-login-config>
4.     <form-login-page>/login.html</form-login-page>
5.     <form-error-page>/error.html</form-error-page>
6.   </form-login-config>
7. </login-config>
```

OWASP
Open Web Application Security Project

Authorization Annotations

roles permitted to execute operation

```
@Path("/customers")
@RolesAllowed({"ADMIN", "CUSTOMER"})
public class CustomerResource {
    @GET
    @Path("{id}")
    @Produces("application/xml")
    public Customer getCustomer(@PathParam("id")
          int id) {...}
    @RolesAllowed("ADMIN")
    @POST
    @Consumes("application/xml")
    public void createCustomer(Customer cust) {...}
    @PermitAll
    @GET
    @Produces("application/xml")
    public Customer[] getCustomers() {}
}
```

any authenticated user

OWASP
Open Web Application Security Project

JAX-RS Security Context

```
public interface SecurityContext {
```

Determine the identity of the user

```
   public Principal getUserPrincipal();
```

check whether user belongs to a certain role

```
   public boolean isUserInRole(String role);
```

whether this request was made using a secure channel

```
   public boolean isSecure();

   public String getAuthenticationScheme();
}
```

OWASP
Open Web Application Security Project

JAX-RS Security Context

```
@Path("/customers")
public class CustomerService {
   @GET
   @Produces("application/xml")
   public Customer[] getCustomers(@Context
      SecurityContext sec) {
      if (sec.isSecure() && !sec.isUserInRole("ADMIN")){
        logger.log(sec.getUserPrincipal() +
                        " accessed customer database.");
      }
      ...
   }
}
```

check whether user belongs to a certain role

Determine the identity of the user

OWASP
Open Web Application Security Project

Anti CSRF

For resources exposed by RESTful web services, it's important to make sure any PUT, POST and DELETE request is protected from Cross Site Request Forgery.

- **Custom request header with token INTERNAL_TOKEN:{token value}**

OWASP
Open Web Application Security Project

Insecure Direct Object Reference

Suppose there is a bank web service as below – what will happen?

- https://example.com/account/325365436/transfer?amount=$100.00&toAccount=473846376

OWASP
Open Web Application Security Project

Output encoding

- X-Content-Type-Options: nosniff
- JSON encoding

OWASP
Open Web Application Security Project

Chapter 3
Some attack examples

OWASP
Open Web Application Security Project

Case 1 - XXE – XML External Entity

Input

```
<?xml version="1.0"?>
<!DOCTYPE customer[<!ENTITY name SYSTEM "file:///c:/temp/password">]>
<customer>
  <name>&name;</name>
</customer>
```

OWASP
Open Web Application Security Project

Output

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<customer>

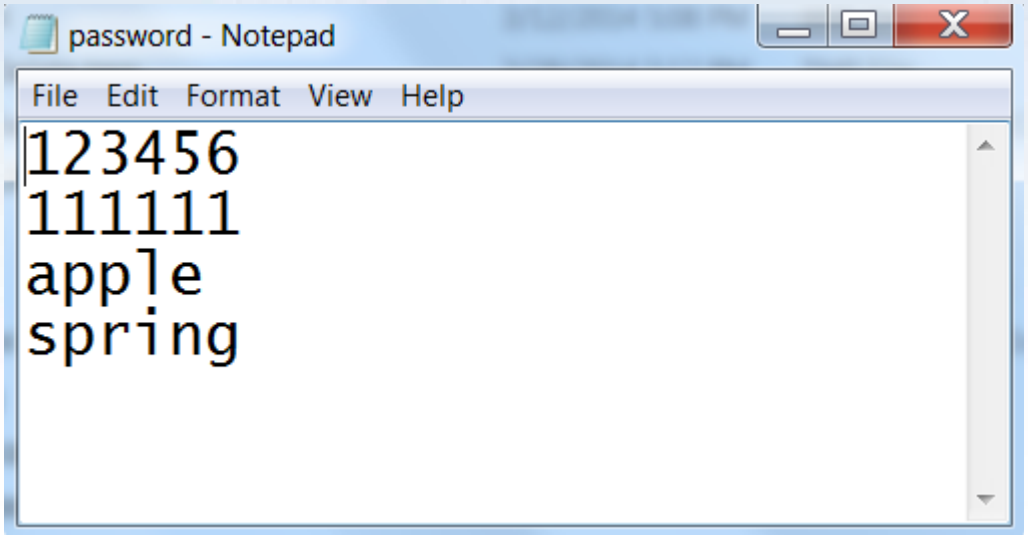
<name>123456

111111

apple

spring</name>

</customer>



OWASP
Open Web Application Security Project

XXE - Solution

```
XMLInputFactory xif = XMLInputFactory.newFactory();

xif.setProperty(XMLInputFactory.IS_SUPPORTING_EXTERNAL_
ENTITIES, false);

xif.setProperty(XMLInputFactory.SUPPORT_DTD, false);
```

Case 2 - HPPP - HTTP Path & Parameter Pollution

User

- Hacker

Security Filter/ Servlet

- Allows GET requests for public but POST, PUT and DELETE for only admin users
- /creditInfo

REST Service

- Provides credit info

OWASP
Open Web Application Security Project

Bypass

User

- Hacker
- "_method" parameter

Security Filter/ Servlet

- Looks like a GET **but turns into PUT, POST, or DELETE**
- creditInfo?_method=PUT

REST Service

- Updates credit info

Think by yourself

```
String entity = request.getParameter("entity");
String id = request.getParameter("id");
URL urlGET = new URL("http://svr.com:5984/client/" + entity + "?id=" + id );
```

Change it to a POST to the following URL

http://svr.com:5984/admin

Answer

```
String entity = request.getParameter("entity");

String id = request.getParameter("id");

URL urlGET = new URL("http://svr.com:5984/client/" + "../admin" + "?id=" +
"1&_method=POST" );
```

Change it to a POST to the following URL

http://svr.com:5984/admin

OWASP
Open Web Application Security Project

Case 3 - XML serialization issues

- REST API allows the raw input of XML to be converted to native objects. This deserialization process can be used to execute arbitrary code on the REST server.

XMLDecoder in RESTlet

org.restlet.representation

Class ObjectRepresentation<T extends Serializable>

```
java.lang.Object
  └org.restlet.representation.Variant
      └org.restlet.representation.RepresentationInfo
          └org.restlet.representation.Representation
              └org.restlet.representation.StreamRepresentation
                  └org.restlet.representation.OutputRepresentation
                      └org.restlet.representation.ObjectRepresentation<T>
```

Type Parameters:

`T` - The class to serialize, see `Serializable`

```
public class ObjectRepresentation<T extends Serializable>
extends OutputRepresentation
```

Representation based on a serializable Java object.
It supports binary representations of JavaBeans using the `ObjectInputStream` and `ObjectOutputStream` classes. In this case, it handles representations having the following media type: `MediaType.APPLICATION_JAVA_OBJECT` ("application/x-java-serialized-object"). It also supports textual representations of JavaBeans using the `XMLEncoder` and `XMLDecoder` classes. In this case, it handles representations having the following media type: `MediaType.APPLICATION_JAVA_OBJECT_XML` ("application/x-java-serialized-object+xml").

XMLDecoder Demo 1 - normal

XMLDecoder Demo 2 - Calculator

XMLDecoder Demo 3 – Get Shell via MetaSploit

Share – Learn - Secure

If you want to share something with us in the next OWASP session, just give me an email: shanda.wang@owasp.org.cn

OWASP
Open Web Application Security Project

Thank you

OWASP
Open Web Application
Security Project