Development of the Security Framework based on OWASP ESAPI for JSF2.0

Autor Matthey Samuel

Website http://www.security4web.ch

14 May 2013

1. Introduction

This project is a continuation of the bachelor thesis (Master Thesis - Applied Computer Science Albert-Ludwigs-Universität Freiburg im Breisgau - "Development of the Security Framework based on OWASP ESAPI for JSF2.0" by Rakeshkumar Kachhadiya) created in May 2012. The goal is to improve more security on the 'File based authorization' module.

2. File based authorization module

This module gives permissions to a specific user to visualize some areas or pages on the presentation layer.

It's responsible to maintain the user information in the file with their assigned roles but also setting the rendering components false if the accessible user tries to retrieve the page.

2.1 Presentation

We have at the left of the Figure 2.1, the 'JSF Framework', it is represented by a cycle which is composed of six steps. A request is initiated by the 'Restore view' step and completed generally as response by the 'Render Response' step.

We note that the 'File Based Authorization' module interacts with the fourth stage 'Update model values' of the JSF cycle.
At this step, the component tree that makes up the page is already created! The responsible step that creates the three is the 'Restore view' step. This tree is stored in a FacesContext object type and will be used throughout the processing of the request.

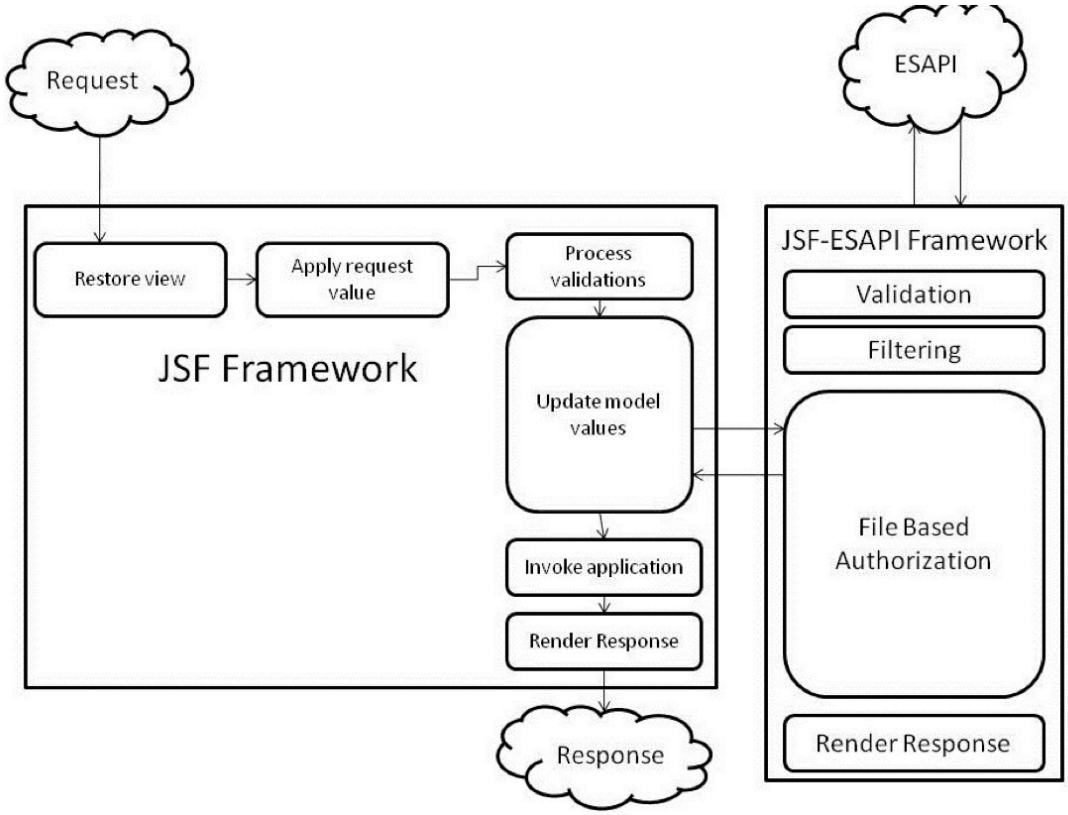

Figure 2.1 File Based Authorization Module {1}

3. Enhance the security

The aim is to achieve this ‘File based authorization’ before the tree is created.

To achieve this, two solutions are possible. The first is to create a ‘ JSF validator’ that will be executed by the ‘Restore view’ step, before the creation of the tree. The second solution is to create a Servlet Filter.

3.1 FaceServlet controller

The FaceServlet controller treats the request before entering into the JSF Lifecycle.

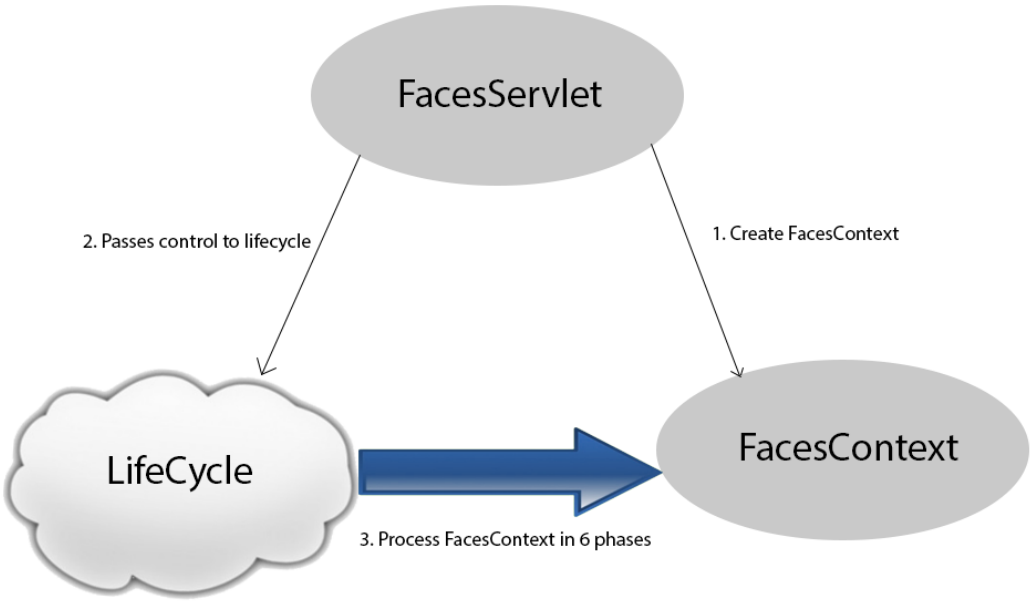

Figure 1.2 FacesServlet

FacesServlet creates a FacesContext object, which contains the ServletContext, ServletRequest, and the ServletResponse Object. The web container passes that to a service method. Then, the FacesServlet hands over control to the LifeCylce Object and process the FacesContext!

This solution filter all '.xhtml' file for example and if the user is not logged or not authorized to view the content of the page or a complete page, he's redirect to an error!

3.2 Attack scenario

When the request enters in the JSF cycle, the tree of all components is created! When this request is completed by the 'Render Response' phase, the component tree is now rendered as HTML!

At this moment, an attacker can easily know the different node ID's available in the source code of the page. He can now generated a POST request with this ID that allow to manually access this page or a area of the page, yet, it is not allowed to view this content!

3.3 Protect this attack

To secure this attack, we create a recursive function that creates a blacklist containing all ID's from every node of the tree. This list will then be inserted into the session!

```
// function to add all node id's in a blacklist
```

```
protected static void disableRec(UIComponent c,List<String> blackList){

        System.out.println("disable"+c.getClientId());
        blackList.add(c.getClientId());
        children = c.getChildren();

        for(UIComponent child : children){
                disableRec(child,blackList);
        }
  }
```

When the attacker generates the POST request, we verify that the ID entered in this request is on the blacklist or not. If the ID is available on the blacklist (that means that he certainly try to modify the tree node), the POST request should not be executed and an error will be generated.

A servlet filter needs to be implemented, which will be executed before the tree was rendered! This filter simply retrieves the blacklist stored in the session and starts the verification.

EsapiPOSTFilter

```
public class EsapiPOSTFilter implements Filter {

        private FilterConfig fc;
        List<UIComponent> children;

        @Override
        public void init(FilterConfig filterConfig) throws ServletException {
                // TODO Auto-generated method stub
                this.fc = filterConfig;
        }

        @Override
        public void destroy() {
                // TODO Auto-generated method stub

                this.fc = null;
        }

        @Override
        public void doFilter(ServletRequest request, ServletResponse response,
                        FilterChain filterChain) throws IOException, ServletException {
                // TODO Auto-generated method stub


                HttpServletRequest req = (HttpServletRequest)request;
                HttpServletResponse res = (HttpServletResponse)response;

                // Get blacklist in session
                List<String> blackList = (List<String>)
req.getSession().getAttribute("blackList");
```

```
                    //print the blacklist id different as null
                    if(blackList != null){
                            for(String black : blackList){
                                    System.out.print(black+", ");
                            }
                    }

                    if(blackList==null){
                            blackList = new LinkedList<String>();
                    }

                    Enumeration<String> enumParam = req.getParameterNames();
               while(enumParam.hasMoreElements()){

                    String id = enumParam.nextElement();

               // if the ID is in the blacklist send error
               if ( blackList.contains(id) ) {

          res.sendError(javax.servlet.http.HttpServletResponse.SC_UNAUTHORIZED);
                            return;
                    }
               }

               // clear the blacklist
               blackList.clear();
               req.setAttribute("blackList", blackList);
          filterChain.doFilter(req, res);
          }

}
```

This blacklist should be emptied once the filtering of the POST request is completed.

Web.xml

To activate the filter in this project, we need to add this part of code in the 'web.xml' file available in the 'Web-INF' folder.

```
<filter>
        <filter-name>POSTFilter</filter-name>
        <filter-class>ch.security4web.esapi.authentication.EsapiPOSTFilter</filter-class>
</filter>

<filter-mapping>
        <filter-name>POSTFilter</filter-name> <url-pattern>*.xhtml</url-pattern>
</filter-mapping>
```

3.4 Try to achieve the attack (with Webscarab)

3.4.1 Installation

First, install the Webscarab java application in OWASP web site:

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

3.4.2 Webscarab configuration

Webscarab is used to intercepts POST requests, and also used to modify the request.
We need to configure it by clicking in the 'intercept request' in the Proxy tab. See Figure 3.4.2

Figure 3.4.2

3.4.3 Practice

Consider a simple login with username and passwords 'Figure 3.4.3', two types of users are available: administrators and users who each have access to certain areas.

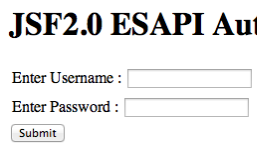

Figure 2.4.3

Administrators have access to the general and admin panels. The users only have access to the general and user panels.
An input field has been added in the admin and user zone. And finally a submit button.

We need to log in with user or admin account, in this case it is an admin account and we need to add a value (tryToHack) in the input field 'Figure 3.4.3.2'

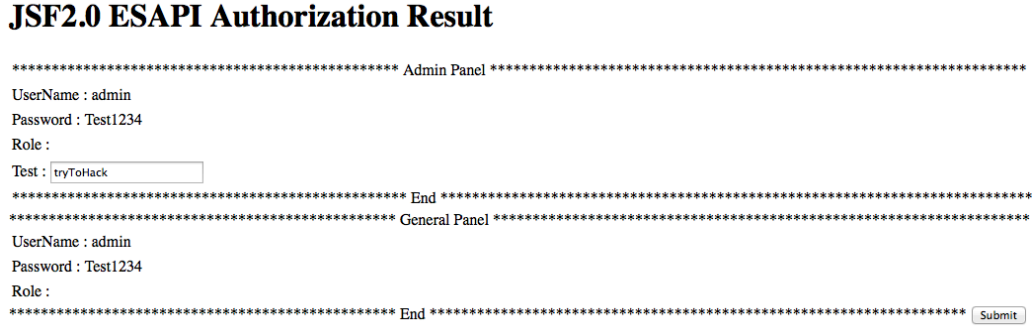

Figure 3.4.3.2

By clicking on the 'submit' button, Webscarab intercepts the request (Figure 3.4.3.3) and we note that the id of the input which we inserted the "tryToHack" value is "j_id_5%3Atest". We will replace this id, with the id of the input of the user panel.

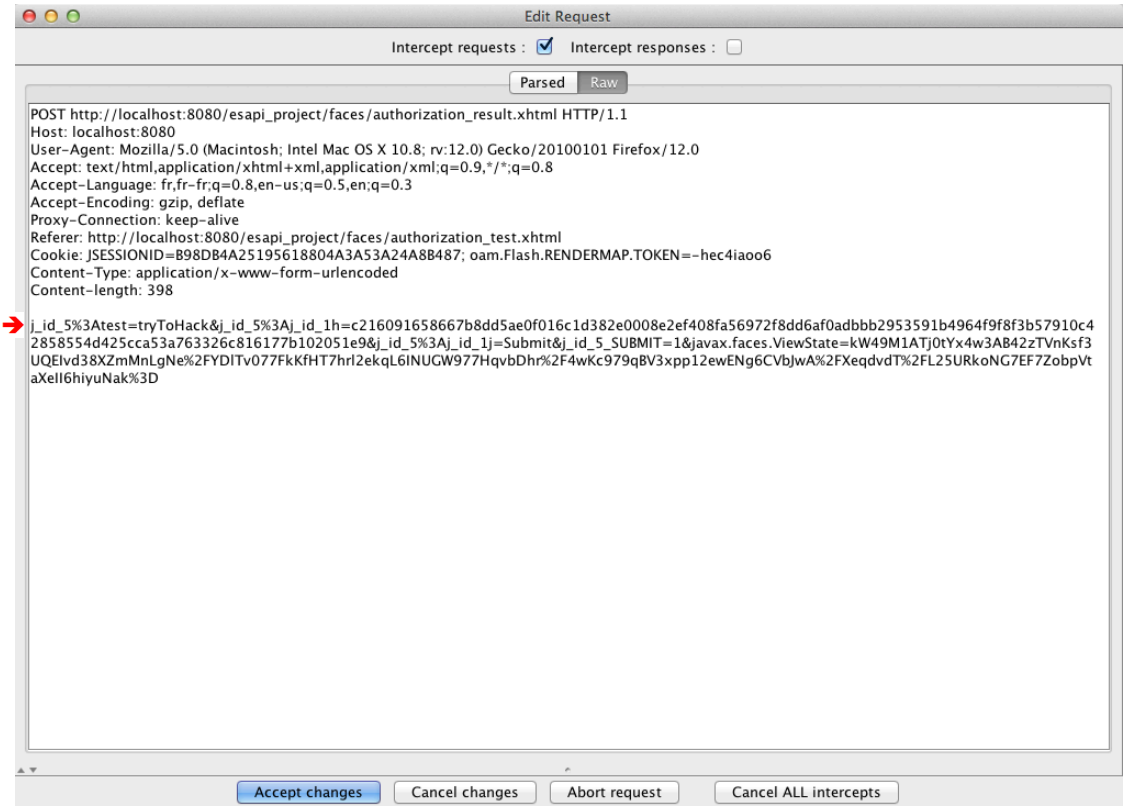

Figure 3.4.3.2

Modify the request and enter the input id (test2) of the user panel and send the request.

```
j_id_5%3Atest2=tryToHack&j_id_5%3Aj_id_1h=c216091658667b8dd5ae0f016c1d382e0008e2ef408fa56972f8dd6af0adbbb2953591b4964f9f8f3b57910c
```

The application has detected the id 'test2' in the blacklist and the filter redirects you to an error page!

4. Bibliography

- Master Thesis - Applied Computer Science Albert-Ludwigs-Universität Freiburg im Breisgau - "**Development of the Security Framework based on OWASP ESAPI for JSF2.0**" *by Rakeshkumar Kachhadiya 2 May 2012, {1}*

- https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API