CODE WHITE

FINEST HACKING

MAY 30, 2018

Poor RichFaces

RichFaces is one of the most popular component libraries for JavaServer Faces (JSF). In the past, two vulnerabilities (CVE-2013-2165 and CVE-2015-0279) have been found that allow RCE in versions 3.x ≤ 3.3.3 and 4.x ≤ 4.5.3. Code White discovered two new vulnerabilities which bypass the implemented mitigations. Thereby, all RichFaces versions including the latest 3.3.4 and 4.5.17 are vulnerable to RCE.

INTRODUCTION

JavaServer Faces (JSF) is a framework for building user interfaces for web applications. While there are only two major JSF implementations (i. e., Apache MyFaces and Oracle Mojarra), there are several component libraries, which provide additional UI components and features. RichFaces is one of the most popular libraries among these component libraries and since it became part of JBoss (and thereby also part of Red Hat), it is also part of several JBoss/Red Hat products, for example JBoss EAP and JBoss Portal.[1]

RichFaces has three major version branches: 3.x, 4.x, and 5.x. However, as 5.x has never left alpha state, it is rather irrelevant. In early 2016, the developers of RichFaces announced the end-of-life of RichFaces in June 2016. The latest releases of the respective branches are 3.3.4 and 4.5.17.

THE PAST

In the past, two significant vulnerabilities have been discovered by Takeshi Terada of MBSD, which both affect various RichFaces versions:

CVE-2013-2165: Arbitrary Java Deserialization in RichFaces 3.x ≤ 3.3.3 and 4.x ≤ 4.3.2
: Deserialization of arbitrary Java serialized object streams in *org.ajax4jsf.resource.ResourceBuilderImpl* allows remote code execution.

CVE-2015-0279: Arbitrary EL Evaluation in RichFaces 4.x ≤ 4.5.3 (RF-13977)
: Injection of arbitrary EL method expressions in *org.richfaces.resource.MediaOutputResource* allows remote code execution.

Both vulnerabilities rely on the feature to generate images, video, sounds, and other resources on the fly based on data provided in the request. The provided data is either interpreted as a plain array of bytes or as a Java serialized object stream. In RichFaces 3.x, the data gets appended to the URL path preceded by either `/DATB/` (byte array) or `/DATA/` (Java serialized object stream); in RichFaces 4.x, the data is transmitted in a request parameter named `db` (byte array) or `do` (Java serialized object stream). In all cases, the binary data is compressed using DEFLATE and then encoded using a URL-safe Base64 encoding.

CVE-2013-2165: Arbitrary Java Deserialization

This vulnerability is a straight forward Java deserialization vulnerability. When a RichFaces 3.x resource is requested, it eventually gets processed by `ResourceBuilderImpl.getResourceDataForKey(String)`. If the requested resource key begins with `/DATA/`, the remaining data gets decoded and decompressed (using `ResourceBuilderImpl.decrypt(byte[])`, which actually, despite its name, does not incorporate encryption[2]) and finally deserialized without any further validation.

In RichFaces 4.x, it is basically the same: the *org.richfaces.resource.DefaultCodecResourceRequestData* holds the request data passed via `db`/`do` and `Util.decodeObjectData(String)` is used in the latter case. That method then decodes and decompresses the data in a similar way and finally deserializes it without any further validation.

This can be exploited with *ysoserial* using a suitable gadget.

The arbitrary Java deserialization was patched in RichFaces 3.3.4 and 4.3.3 by introducing look-ahead deserialization with a limited set of whitelisted classes.[3] Due to several aftereffects, the list was extended occasionally.[4]

CVE-2015-0279: Arbitrary EL Evaluation

The RichFaces issue RF-13977 corresponding to this vulnerability is public and actually quite detailed. It describes that the RichFaces Showcase application utilizes the *MediaOutputResource* dynamic resource builder. The data object passed in the `do` URL parameter holds the state object, which is used by `MediaOutputResource.restoreState(FacesContext, Object)` to restore its state. This includes the *contentProducer* field, which is expected to be a *MethodExpression* object. That *MethodExpression* later gets invoked by

`MediaOutputResource.encode(FacesContext)` to pass execution to the referenced method to generate the resource's contents. In the mentioned example, the EL method expression `#{mediaBean.process}` references the *process* method of a Java Bean named *mediaBean*.

Now the problem with that is that the EL expression can be changed, even just with basic Linux utilities. There is no protection in place that would prevent one from tampering with it. Depending on the EL implementation, this allows arbitrary code execution, as demonstrated by the reporter:

```
#{request.getClass().getClassLoader().loadClass("java.lang.Runtime").getMethod("getRuntime
```

methodexpression.el hosted with ❤ by **GitHub** **view raw**

However, exploitation of this vulnerability is not always that easy. Especially if there is no existing sample of a valid `do` state object that can be tampered with. Because if one would want to create the state object, it would require the use of compatible libraries, otherwise the deserialization may fail. Moreover, the EL implementation does not allow arbitrary expressions with parameterized invocations in method expressions as this has only just been added in EL 2.2. EL exploitation is quite an interesting topic in itself.

The patch for this issue introduced in RichFaces 4.5.4 was to check the expression of the *contentProducer* whether it contains a parenthesis. This would prevent the invocation of methods with parameters like `loadClass("java.lang.Runtime")`.

THE PRESENT

The kind of the past vulnerabilities led to the assumption that there may be a way to bypass the mitigations. And after some research, two ways were found to gain remote code execution in a similar manner also affecting the latest RichFaces versions 3.3.4 and 4.5.17:

RF-14310: Arbitrary EL Evaluation in RichFaces 3.x ≤ 3.3.4

Injection of arbitrary EL expressions allows remote code execution via *org.richfaces.renderkit.html.Paint2DResource*.

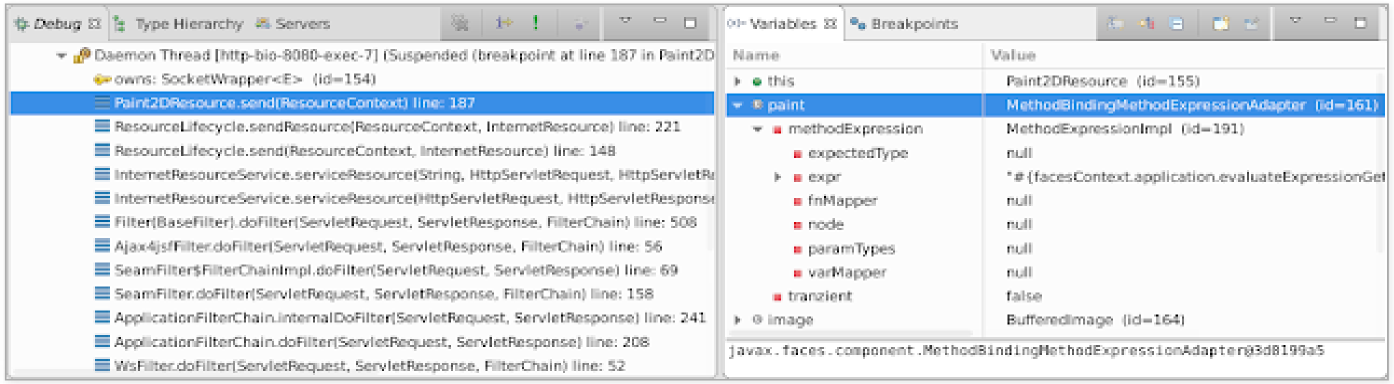
RF-14309: Arbitrary EL Evaluation in RichFaces 4.5.3 ≤ 4.5.17

Injection of arbitrary EL variable mapper allows to bypass mitigation of CVE-2015-0279 and thereby remote code execution.

Although the issues RF-14309 and RF-14310 were discovered in the order of their identifier, we'll explain them in the opposite order. Also note that the issues are not public but only visible to persons responsible to resolve security issues.

RF-14310: Arbitrary EL Evaluation

This vulnerability is very similar to CVE-2015-0279/RF-13799. While the injection of arbitrary EL expressions was possible right from the beginning, there is always a need to get them triggered somehow. This similarity was found in the org.richfaces.renderkit.html.Paint2DResource class. When a resource of that type gets requested, its `send(ResourceContext)` method gets called. The resource data transmitted in the request must be an *org.richfaces.renderkit.html.Paint2DResource$ImageData* object. This passes the whitelisting as *ImageData* extends *org.ajax4jsf.resource.SerializableResource*, which actually was introduced in 3.3.4 to fix the Java deserialization vulnerability.



```
Daemon Thread [http-bio-8080-exec-7] (Suspended (breakpoint at line 187 in Paint2D
  owns: SocketWrapper<E>  (id=154)
  Paint2DResource.send(ResourceContext) line: 187
  ResourceLifecycle.sendResource(ResourceContext, InternetResource) line: 221
  ResourceLifecycle.send(ResourceContext, InternetResource) line: 148
  InternetResourceService.serviceResource(String, HttpServletRequest, HttpServletRe
  InternetResourceService.serviceResource(HttpServletRequest, HttpServletResponse
  Filter(BaseFilter).doFilter(ServletRequest, ServletResponse, FilterChain) line: 508
  Ajax4jsfFilter.doFilter(ServletRequest, ServletResponse, FilterChain) line: 56
  SeamFilter$FilterChainImpl.doFilter(ServletRequest, ServletResponse) line: 69
  SeamFilter.doFilter(ServletRequest, ServletResponse, FilterChain) line: 158
  ApplicationFilterChain.internalDoFilter(ServletRequest, ServletResponse) line: 241
  ApplicationFilterChain.doFilter(ServletRequest, ServletResponse) line: 208
  WsFilter.doFilter(ServletRequest, ServletResponse, FilterChain) line: 52
```

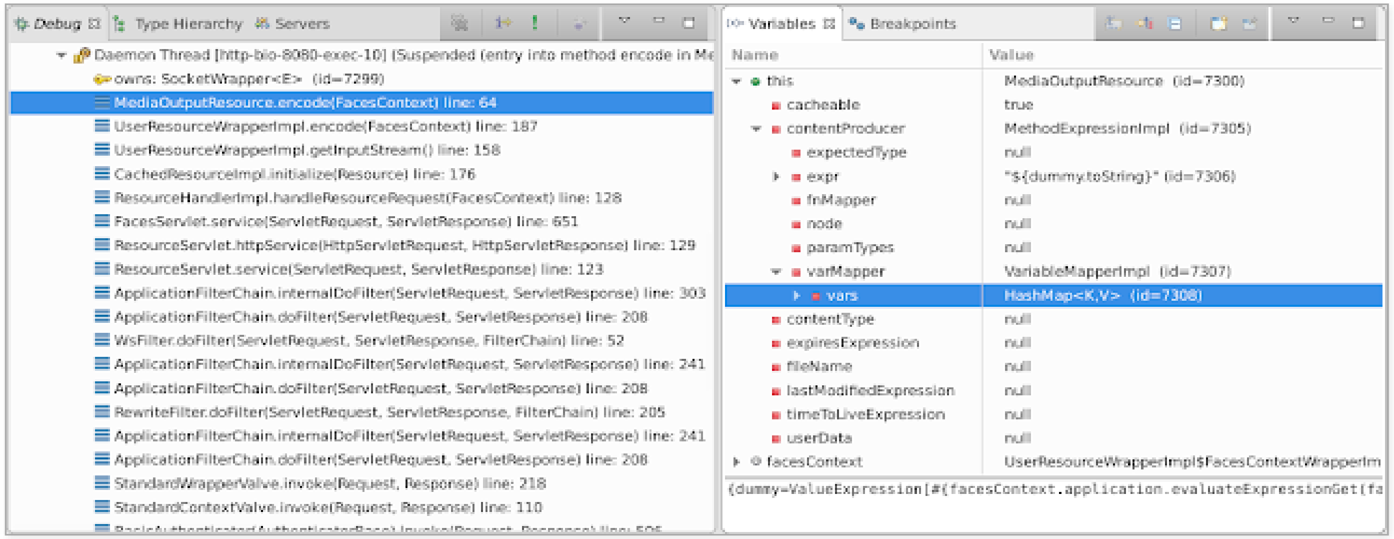
Name	Value
this	Paint2DResource (id=155)
paint	MethodBindingMethodExpressionAdapter (id=161)
methodExpression	MethodExpressionImpl (id=191)
expectedType	null
expr	"#{facesContext.application.evaluateExpressionGet
fnMapper	null
node	null
paramTypes	null
varMapper	null
tranzient	false
image	BufferedImage (id=164)

```
javax.faces.component.MethodBindingMethodExpressionAdapter@3d8199a5
```

RF-14309: Arbitrary EL Evaluation

As the patch to CVE-2015-0279 introduced in 4.5.4 disallowed the use of parenthesis in the EL method expression of the *contentProducer*, it seemed like a dead end. But if you are fimilar with EL internals, you would know that they can have custom function mappers and variable mappers, which are used by the *ELResolver* to resolve functions (i. e., *name* in `${prefix:name()}`) and variables (i. e., *var* in `${var.property}`) to *Method* and *ValueExpression* instances respectively. Fortunately, various *VariableMapper* implementations were added to the whitelist starting with 4.5.3.[5]

So to exploit this, all that is needed is to use a variable in the *contentProducer* method expression like `${dummy.toString}` and add an appropriate *VariableMapper* to the method expression that maps `dummy` to a *ValueExpression* of your choice.



```
Daemon Thread [http-bio-8080-exec-10] (Suspended (entry into method encode in Me
  owns: SocketWrapper<E>  (id=7299)
  MediaOutputResource.encode(FacesContext) line: 64
  UserResourceWrapperImpl.encode(FacesContext) line: 187
  UserResourceWrapperImpl.getInputStream() line: 158
  CachedResourceImpl.initialize(Resource) line: 176
  ResourceHandlerImpl.handleResourceRequest(FacesContext) line: 128
  FacesServlet.service(ServletRequest, ServletResponse) line: 651
  ResourceServlet.httpService(HttpServletRequest, HttpServletResponse) line: 129
  ResourceServlet.service(ServletRequest, ServletResponse) line: 123
  ApplicationFilterChain.internalDoFilter(ServletRequest, ServletResponse) line: 303
  ApplicationFilterChain.doFilter(ServletRequest, ServletResponse) line: 208
  WsFilter.doFilter(ServletRequest, ServletResponse, FilterChain) line: 52
  ApplicationFilterChain.internalDoFilter(ServletRequest, ServletResponse) line: 241
  ApplicationFilterChain.doFilter(ServletRequest, ServletResponse) line: 208
  RewriteFilter.doFilter(ServletRequest, ServletResponse, FilterChain) line: 205
  ApplicationFilterChain.internalDoFilter(ServletRequest, ServletResponse) line: 241
  ApplicationFilterChain.doFilter(ServletRequest, ServletResponse) line: 208
  StandardWrapperValve.invoke(Request, Response) line: 218
  StandardContextValve.invoke(Request, Response) line: 110
```

Name	Value
this	MediaOutputResource (id=7300)
cacheable	true
contentProducer	MethodExpressionImpl (id=7305)
expectedType	null
expr	"${dummy.toString}" (id=7306)
fnMapper	null
node	null
paramTypes	null
varMapper	VariableMapperImpl (id=7307)
vars	HashMap<K,V> (id=7308)
contentType	null
expiresExpression	null
fileName	null
lastModifiedExpression	null
timeToLiveExpression	null
userData	null
facesContext	UserResourceWrapperImpl$FacesContextWrapperIm

```
{dummy=ValueExpression[#{facesContext.application.evaluateExpressionGet(fa
```

THE FUTURE

RichFaces has reached its end-of-life in June 2016 and their RichFaces End-Of-Life Questions & Answers is pretty clear on the time thereafter:

> **What will happen if a serious bug or security issue is discovered in the future?**
>
> There will be no patches after the end of support. In case of discovering a serious issue you will have to develop a patch yourself or switch to another framework.
>
> – RichFaces End-Of-Life Questions & Answers

So we can't expect official patches.

THE BONUS

While looking for ways to exploit the recent versions of RichFaces, there were two classes in the JSF API 2.0 and later, which seemed promising:

- *javax.faces.component.ValueBindingValueExpressionAdapter*
- *javax.faces.component.ValueExpressionValueBindingAdapter*

The interesting thing about these classes is that they have a `equals(Object)` method, which eventually calls `getType(ELContext)` on a EL value expression. For example, if `equals(Object)` gets called on a *ValueExpressionValueBindingAdapter* object with a *ValueExpression* object as `other`, `getType(ELContext)` of `other` gets called. And as the value expression has to be evaluated to determine its resulting type, this can be used as a Java deserialization primitive to execute EL value expressions on deserialization.

```
// classes of EL implementation
javax.el.ValueExpression valueExpressionImpl1 = new org.apache.el.ValueExpressionImpl("#{
javax.el.ValueExpression valueExpressionImpl2 = new org.apache.el.ValueExpressionImpl("#{

```

```java
// classes of EL API/JSF API only
javax.faces.el.ValueBinding valueBindingValueExpressionAdapter = new javax.faces.componen
javax.el.ValueExpression valueExpressionValueBindingAdapter = new javax.faces.component.V

// test valueExpressionValueBindingAdapter.equals(valueExpressionImpl2) using HashMap
java.util.Map hashMap = new java.util.HashMap();
hashMap.put(valueExpressionImpl2, "");
hashMap.put(valueExpressionValueBindingAdapter, "");
```

jsf-gadget.java hosted with ❤ by **GitHub** **view raw**

This is very similar to the *Myfaces1* and *Myfaces2* gadgets in *ysoserial*.[6] However, while they require Apache MyFaces, this one is independent from the JSF implementation and only requires a matching EL implementation.

Unfortunately, this gadget does not work for RichFaces. The reason for that is that *ValueExpressionValueBindingAdapter* needs to have a valid value binding as `getType(ELContext)` gets called first. But *javax.faces.el.ValueBinding* is not whitelisted. And wrapping it in a *StateHolderSaver* does not work because the state object is of type *Object[]* and therefore the cast to *Serializable[]* in `StateHolderSaver.restore(FacesContext)` fails.[7] This is probably a bug in RichFaces as *Serializable[]* is not whitelisted either although *StateHolderSaver* uses *Serializable[]* internally on *StateHolder* instances.

CONCLUSION

It has been shown that all RichFaces versions 3.x and 4.x including the latest 3.3.4 and 4.5.17 are exploitable by one or multiple vulnerabilities:

- RichFaces 3

    - 3.1.0 ≤ 3.3.3: CVE-2013-2165
    - 3.1.0 ≤ 3.3.4: RF-14310
- RichFaces 4
    - 4.0.0 ≤ 4.3.2: CVE-2013-2165
    - 4.0.0 ≤ 4.5.4: CVE-2015-0279
    - 4.5.3 ≤ 4.5.17: RF-14309

As we can't expect official patches, one way to mitigate all these vulnerabilities is to block requests to the concerned URLs:

- Blocking requests of URLs with paths containing `/DATA/` should mitigate CVE-2013-2165 and RF-14310.
- Blocking requests of URLs with paths containing `org.richfaces.resource.MediaOutputResource` (literally or URL encoded) should mitigate CVE-2015-0279 and RF-14309.

- [1] See JBoss Enterprise Application Platform Component Details and Red Hat JBoss Portal Component Details for further version information.
- [2] However, *org.ajax4jsf.util.base64.Codec* does support DES encryption if a password is set.
- [3] See commit richfaces4/core@12ee116
- [4] See the history of *org/richfaces/resource/resource-serialization.properties* in 4.x ≤ 4.3.x and in 4.5.x for reference.
- [5] *com.sun.el.lang.VariableMapperImpl* and *org.apache.el.lang.VariableMapperImpl* were added in 4.5.3, *org.jboss.el.lang.VariableMapperImpl* was added in 4.5.8; see also the history of *org/richfaces/resource/resource-serialization.properties* in 4.5.x
- [6] See Myfaces1.java and Myfaces2.java in the *ysoserial* repository on GitHub.
- [7] This actually depends on the JSF API implementation and version. For example, *org.jboss.spec.javax.faces/jboss-jsf-api_2.1_spec* and *org.glassfish/javax.faces* do have this unfortunate behavior in all versions while *com.sun.faces/jsf-api* added it in version 2.1.0.

Posted by Markus Wulftange at 3:00 PM

Tags Gadget, Vulnerability Details

FURTHER LINKS

- Homepage
- Twitter
- Xing
- LinkedIn
- Privacy Policy
- Impressum

JOIN US

Career @ Code White

BLOG ARCHIVE

May (1) ▼