某客服服务系统通杀漏洞(逆向SO算法到GETSHELL)

来看看这个东西的案例：



web.xml的效果来看看：

<filter>

<filter-name>AuthCheck</filter-name>

<filter-class>org.jivesoftware.admin.AuthCheckFilter</filter-class>

<init-param>

<param-name>excludes</param-name>

<param-value>login\_bysso.jsp,userregisterforclient.jsp,clientGroupmessagerList.action,panel/,randImages.jsp,softphone/,result,ssoOaUserConfig,ssoOaUserList,mypanel-showurl.jsp,sso-,/login.jsp,index.jsp?logout=true,setup/index.jsp,setup/setup-,.gif,.png,error-serverdown.jsp,client\_showaffiche.action,client\_affiche.action,client\_affiche\_list.action,client\_affiche\_scroll.action,client\_mail.jsp,client\_systemaffiche\_scroll.action,clientaffichesystemscroll.jsp,client-systemaffiche-list.action,client\_issueaffiche.action,exportmessagetofile.jsp,clientregister.action,register\_sucess.jsp,ipphoneList.action,bbsdiscuss/dhtreeForums.jsp,js\_list\_nodes.jsp,grouptreehome-inner.jsp,bindgroup-index-inner.jsp,gjpanel-showurl.jsp,clientBindGroupAffEdit.jsp,clientBindGroupAffDelete.jsp,clientAfficheShow.jsp,mypanel-redirect.jsp,panelextends-registertairan.jsp,zk/show\_personalsch.zul,zk/show\_commonsch.zul,zk/show\_notice.zul,zk/sch\_personal.zul,zk/sch\_common.zul,zk/sch\_notice.zul,user-email-edit.jsp,outlinker-export.jsp,outlinker-loadexport.jsp,outlinerload.action,outlinerload\_star.action,user-register.jsp,user-register.action,zkau1,worksheetmanager.action,webcall\_chat/knowledgeSearchTreeData.jsp,share\_own.jsp,web-client-updatepwd.jsp,userinfo.jsp,userinfo-basic.jsp,userinfo-contact.jsp,userinfo-detail.jsp,ucstarVisitorStatueMain.jsp,emp-detail-info-customershow.jsp,changepassword.jsp,xinda\_onlinebusinesshall.jsp,xinda\_clientsso.jsp,xinda\_call.jsp,xinda\_crmsys.jsp,xinda\_ssologin.jsp,emp-detail-info.jsp,clientaffiche\_manager.jsp,blank.jsp,clientaffichesystemscroll2.jsp,client\_systemaffiche\_scroll.action,tradesys-share.jsp,tradesys-fund.jsp,userclient-card.jsp,analysis\_huaxi.action,logincms.jsp,schedulemsglistclient.action,shedulemsg\_list\_client.jsp,updateuser.action,selectgroupbysession.jsp,ace-select-radio-groups.jsp,selectuserbysession.jsp,system-message-schedule-client.jsp,user-basic-info.jsp,user-contace-info.jsp,user-detail-info.jsp,user-basic-info-emp.jsp,user-contact-info-emp.jsp,user-detail-info-emp.jsp,toedituserbasepage.action,toeditusercontactpage.action,toedituserdetailpage.action,toedituserbasepageemp.action,toeditusercontactpageemp.action,toedituserdetailpageemp.action,service\_vote.jsp,mainSeat.action,ucstarMessage-list-new.jsp,ucstarMessageReceiptMain.jsp,vcstar\_sg/,call.do,</param-value>

</init-param>

</filter>

关键的过滤器就是这个，里面有个excludes，代码不分析了，这里就是说excludes里面的这些文件或者目录下的文件访问是不用验证权限的

剩下的大部分都是sevlet，其中眼睛一亮，看到一处文件上传：

<servlet>

<description>This is the description of my J2EE component</description>

<display-name>This is the display name of my J2EE component</display-name>

<servlet-name>FileUploadServlet</servlet-name>

<servlet-class>qflag.ucstar.file.upload.FileUploadServlet</servlet-class>

</servlet>

这个文件到底是干什么的呢，我高兴的太早了，这个里面内容是被加密的，顿时心情就不好了，继续往下翻

<listener>

<listener-class>qflag.security.listener.WebClassLoaderListener</listener-class>

</listener>

这个进去看看：

public void contextInitialized(ServletContextEvent sce) {

System.out.println("[Begin Init Path]");

this.\_initFilebaseDir((String)null);

this.\_initPath(sce == null?null:sce.getServletContext());

this.\_initFile();

this.\_initUrls();

try {

System.out.println("[Begin Init ClassLoader]");

ClassLoader ex = ClassLoader.getSystemClassLoader();

ClassLoader current = Thread.currentThread().getContextClassLoader();

EncryptedURLClassLoader encryptedCl = EncryptedURLClassLoader.newInstance(this.encryptUrls, current);

encryptedCl.setClassDecrypter(DefaultClassDecrypter.getInstance());

System.out.println("[ENCRYPTCLASSLOADER]-----[load encrypted classes!]");

System.out.println(this.fEncryptedFiles);

encryptedCl.loadAllEncryptedClass(this.fEncryptedFiles, (ClassFilter)null);

Thread.sleep(2000L);

System.out.println("[ENCRYPTCLASSLOADER]-----[load successfull!]");

} catch (Exception var5) {

var5.printStackTrace();

}

从这里可以看出来，这里是类加载的一个解密过程，然后才能顺利的走到后面

继续在往里面翻翻：

DefaultClassDecrypter

public class DefaultClassDecrypter implements IClassDecrypter {

private static DefaultClassDecrypter instance = null;

private ProtectionDomain defaultDomain = null;

static {

if(System.getProperty("os.name").toLowerCase().indexOf("windows") >= 0) {

System.out.println("[windows]");

System.loadLibrary(SecurityConstants.getEncryptFileWin());

} else {

System.out.println("[unix]");

System.load(SecurityConstants.getEncryptFileLin());

}

}

private DefaultClassDecrypter() {

}

public static DefaultClassDecrypter getInstance() {

if(instance == null) {

synchronized(ClassEncrypter.class) {

instance = new DefaultClassDecrypter();

}

}

return instance;

}

private native Class defineEncryptedClass1(ClassLoader var1, String var2, byte[] var3, int var4, int var5, ProtectionDomain var6, String var7);

private native Class defineEncryptedClass2(ClassLoader var1, String var2, ByteBuffer var3, int var4, int var5, ProtectionDomain var6, String var7);

public final Class decryptClass(ClassLoader loader, String name, byte[] b, int off, int len, ProtectionDomain pd, String source) {

name = name.replace('.', '/');

return this.defineEncryptedClass1(loader, name, b, off, len, pd, source);

}

public final Class decryptClassFromNet(ClassLoader loader, String name, ByteBuffer b, int off, int len, ProtectionDomain pd, String source) {

return this.defineEncryptedClass2(loader, name, b, off, len, pd, source);

}

}

SecurityConstants

public class SecurityConstants {

private static String ENCRYPT\_FILE\_WIN = "javaCrypto";

private static String ENCRYPT\_FILE\_LIN = "/usr/lib/javaCrypto.so";

static {

if(System.getProperty("os.name").toLowerCase().indexOf("windows") >= 0) {

System.out.println("库目录:" + System.getProperty("java.library.path"));

}

}

public SecurityConstants() {

}

public static String getEncryptFileWin() {

String theData = System.getProperty("encrypt.file", "");

if(theData != null && theData.trim().length() > 0) {

theData = theData.trim();

System.out.println("encrypt load[windows]:" + theData);

return theData;

} else {

return ENCRYPT\_FILE\_WIN;

}

}

public static String getEncryptFileLin() {

String theData = System.getProperty("encrypt.file", "");

if(theData != null && theData.trim().length() > 0) {

theData = theData.trim();

System.out.println("encrypt load[linux]:" + theData);

return theData;

} else {

return ENCRYPT\_FILE\_LIN;

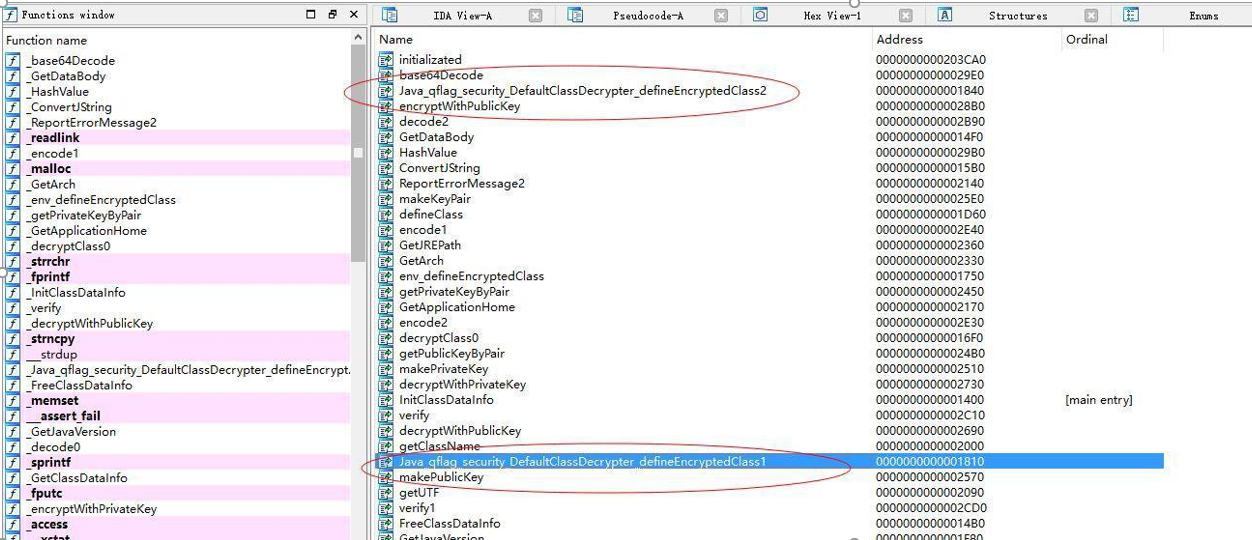
}

}

}

从这里心里就咯噔一下，中间加解密通过so或者dll文件，只能扔到IDA下分析了，高兴的是没有做混淆加密

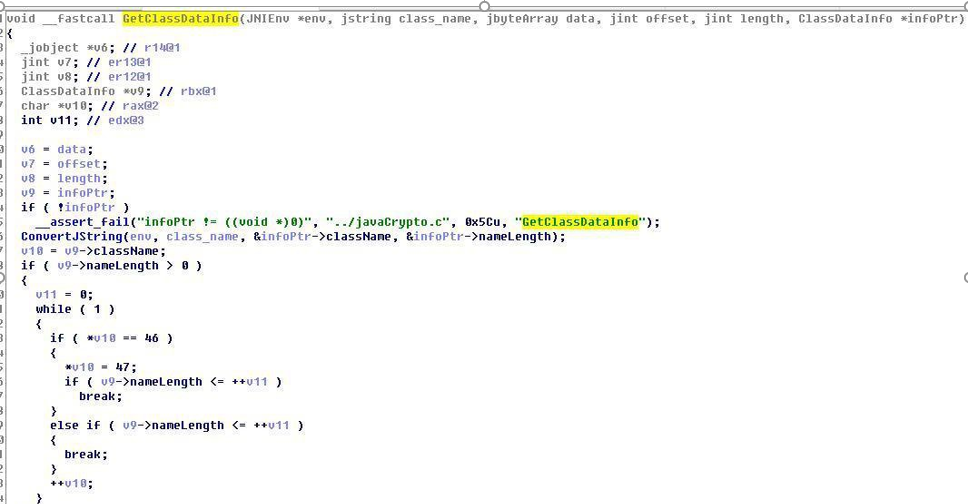
看一下输出表



 这两个就是java JNI调用的接口，分析一下这里面的伪代码看看：

decryptClass0 解密的程序

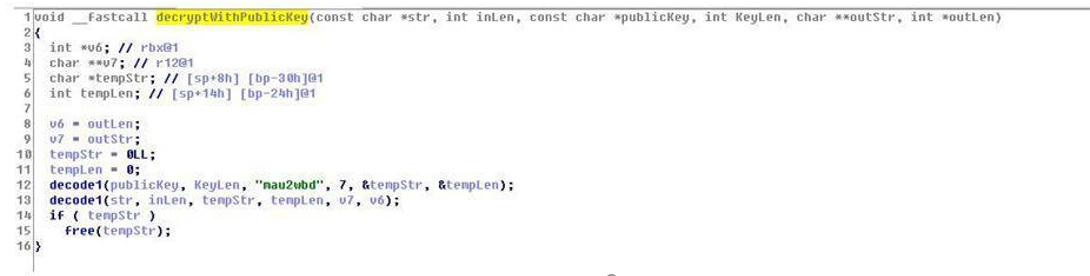




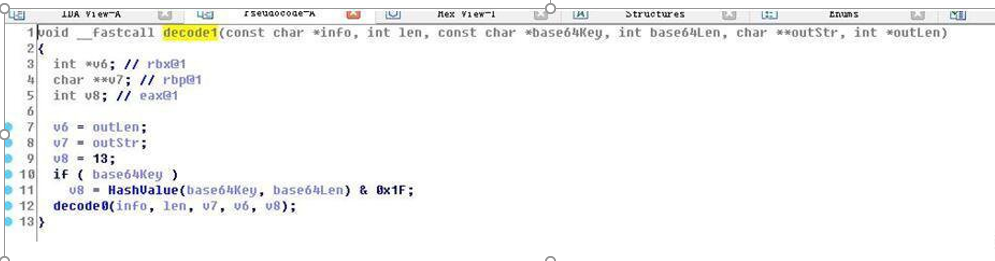
 上面的

GetClassDataInfo这个主要是获取文件的相关信息，并且存到PInfo中

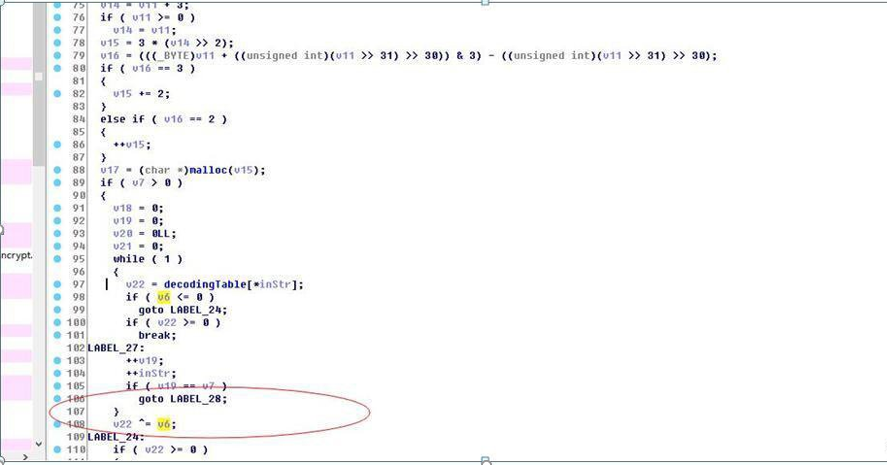
decryptWithPublicKey 这个就是主要的解密函数



第一个是内容，publickey就是刚才的那个classname，decode1的私钥居然写死到了程序里面



这里HashValue的两个值都可控，在看看decode0 调用了base64Decode，直接看这个：



这里的v6就是刚才HashValue算出来的值，这个base64的算法，无非就是根据HashValue生成一个混淆种子，然后跟decoding里面的base64子附近做异或运算

那么剩下的就是classname，根据字面意思我们就当做jar包里面的class名字来看

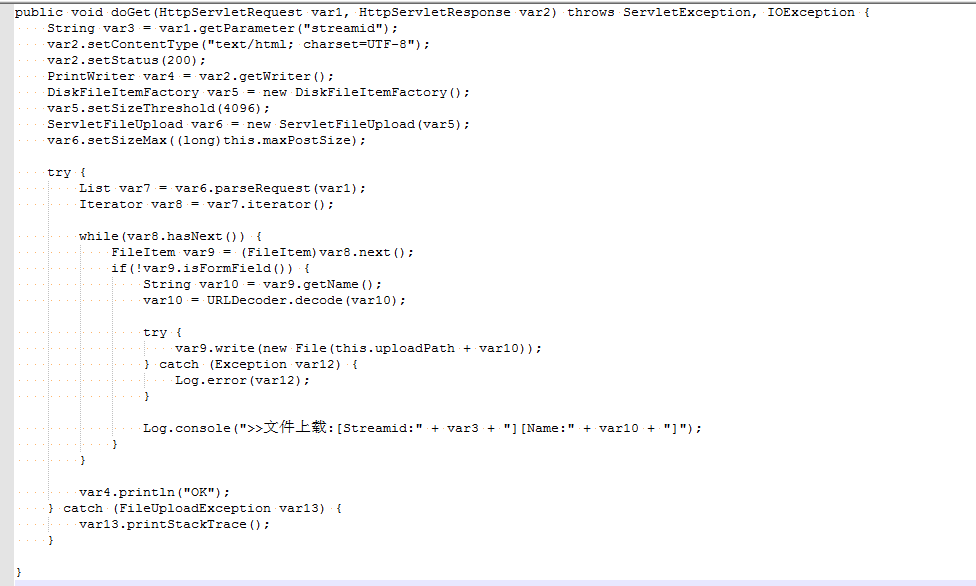
编写C++解密，然后直接去调用这个so，偷个懒

jdecoder.cpp

(linux系统)把javaCrypto.so文件和jdecoder.cpp放到同一个目录下，

编译g++ -o jdec \*.cpp -fPIC -ldl ，然后运行就解密开了，从这里可以推算一件事情，就是说，他自己传进来确实是jar里面的classname，这个程序其实只要这个classname加盐，然后把盐存好，破解成本就很高

解开了，分析漏洞点：



1. List var7 = var6.parseRequest(var1); 这个里面就是初始化文件上传的参数和数据并且解析

2.  var9.write(new File(this.uploadPath + var10));底下直接写文件，看看写入目录在哪里



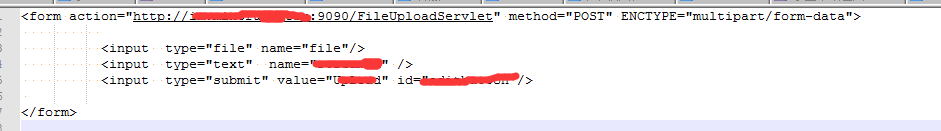
居然不在web目录，看看这套程序的目录结构



安装目录都是固定的，上传文件在bin/res/upload/目录下，不能被直接访问，但是文件明我们可以控制，所以跳跃目录，../../../../web/panel/sysinix123.jsp

这里放到pane1里面就是刚开始说的那个过滤器，这个目录是不用验证权限的

payload：



这里只是举个例子，正确做法就是抓包后，把filename修改为

../../../../web/panel/sysinix123.jsp

访问

