**博文作者：空虚浪子心**

**发布日期：2013-10-14**

**博文内容：**

**摘要**

这篇文章在2012年写成，由于各种原因一直没有发布，当时的情况，是所有的语言底层，都出了补丁，只有java例外，迟迟没有发布。现在发这篇文章出来，仅仅是回顾一种以前的攻击手法，希望大家对于研究漏洞的修补，以及利用方式有所启发。

2011年底，出现了一种新的DOS攻击，Hash碰撞导致的denial of service，影响JAVA、PHP、RUBY等多种语言。我看了漏洞原理，以及包括官方的方案，补丁、源码等，发现解决方案上，有很大的遗漏，他们没有从根本上解决这个漏洞，仅仅是防御了一种“可以直接利用”的手段。

这就意味着，还有各种其他“可以间接利用”的手段。

**正文**

首先看了ben的一片原理分析，传送门这里（PS：看懂“这”篇，才能看懂“这”篇）：

<http://hi.baidu.com/cn_ben/blog/item/d8690ed1809f0891a1ec9c6f.html>

这篇文章，相对于官方的公告，很细节的分析了漏洞原理，甚至漏洞的利用方式，ben没有直接公布利用代码。但是为了让更多的人研究，给大家个提示。之前有人研究过相关技术，有成型代码，只是没有考虑到用在黑客攻击上，所以很容易搜索到一个代码，只要对结果加了正则，就可以直接使用。

有了这篇文章，原理我就不说了，只是说一下其中出彩的地方，本文只说java。Hashmap是java中很常用的类，由于它的结构，会被大面积应用于日常代码中，所以很不幸，tomcat等web服务器在处理用户提交的参数时，直接把参数“键值对”放入了hashmap中。从开发角度上讲，这是非常“教科书”的标准做法。问题在于，java受此漏洞影响，我们却给tomcat打补丁，是因为tomcat直接面向用户访问，最终导致任何人都可以直接攻击web服务器。面对如此严重的情况，tomcat官方自然就坐不住了，迅速出了补丁。

**补丁解读**

看到tomcat修补，我就知道会有更悲惨的事情要发生了，大家（安全工程师们）都会给tomcat打补丁，反而忽略了这其实是java的漏洞。当然，不管它能不能搞定，我们都来看看补丁情况。

Tomcat的补丁，限制用户提交的参数个数。

补丁后的版本tomcat 6.0.35，再次提交攻击POC，已经没有任何影响。这个补丁，针对“tomcat等web服务器在处理用户提交的参数时，直接把参数“键值对”放入了hashmap中”这种情况，做出了修补。

补丁后，又简单的测试了其他可能发生的地方，比如http head、cookie等等。补丁到了这种程度，tomcat已经仁至义尽了，它做到了自己该做的，POC又不能通行，于是大家淡定了。

**JAVA HASH碰撞拒绝服务- - 制作慢性毒药**

这个事情，还没有结束。在开发一个web应用时，有很多地方，都会用到hashmap，这些地方不一定都能够和用户打交道，用户也不一定都能直接控制到。但是这些可以控制的地方，绝对不仅仅是用户post一个东西上来直接打死tomcat，他们还有很多有趣的例子。现在看两个场景，非常的常用。

场景1：录入一个产品

在本地搭建一个数据库，表名为product，列名“ID”、“pname”、“price”。很常见的常用，用户录入一个产品。每次用户提交，都会录入一个产品，比如录入“水果牌MP3”。

场景2：显示所有产品名和价格

|  |
| --- |
| public HashMap getProductMap(){  try {  HashMap hashMap = (HashMap)getSession()  .createSQLQuery("select pname, price from product").setResultTransformer(Transformers.ALIAS\_TO\_ENTITY\_MAP);  return hashMap;  } catch (RuntimeException re) {  throw re;  }  } |

这是hibernate3的一种写法，查询出的数据，直接变为hashmap，把产品名作为hashmap的key。当然，hibernate3只是更加容易触发而已，关键问题是开发人员是如何设计这段程序的，他也可以这样写

|  |
| --- |
| ResultSet rs = db.executeQuery("select pname,price from product");  try {  HashMap hash = new HashMap();  while (rs.next()) {  String name = rs.getString("pname");  String price = rs.getString("price");  hash.put(name,price);  }  } catch (SQLException e) {  e.printStackTrace();  }  db.close(); |

现在，邪恶的攻击者来了，录入大量的产品，这些产品名称的hashcode一致，攻击者可以慢慢的录入，甚至分开几天录入，直到达到可攻击的数据量。当线上服务器执行到这段代码的时候，就会造成DOS。

有趣的是，测试环境的数据，一般不是这样的，这个现象，也许只有在真实环境，才会体现出来。攻击者只管录入数据，他不知道什么时候会产生DOS，他不需要知道。也许会在某个展示页面出现，也许会在后台页面出现，一旦出现，线上服务器CPU直接飙满，一定会让各个相关人员焦头烂额。

我们头脑风暴一下，这样的应用处理，是不是会发生在其他地方呢？

开发人员会有个中稀奇古怪的实现，比如hashmap在session中存放，允许用户加入key-value，或者在application context中存放hashmap等等。攻击者需要做的，只是非常有耐心的，长时间慢慢的录入，一个一个的请求服务器，直到服务器再也“撑不下”。这样不必拘泥于一个post就要打死服务器，才是真正的，令防火墙万分头疼的慢性毒药。

**JAVA HASH碰撞拒绝服务- - JSON OBJECT拒绝服务**

也许上一个例子大家会有异议，因为总是会有开发会说“我怎么可能这样写代码？”。我们切换场景。Web2.0时代（我也不想扯这面大旗，实在太俗了），大家都用ajax，使用json传输。Java对于json有很多种实现，最最常用的，莫过于jsonObject。

JsonObject在解析json字符串时，使用了hashmap存储，所以间接地导致了漏洞的产生。

看这样一段最常用，不得不用的代码，将一段string，变为json对象。

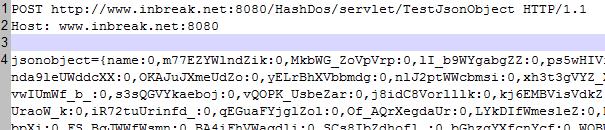
|  |
| --- |
| String json = "{name: 'kxlzx'}";  **try** {  JSONObject jsonObject = **new** JSONObject(json);  System.*out*.println(jsonObject.get("name"));  } **catch** (JSONException e) {  e.printStackTrace();  } |

再深入一下，我们干脆写个servlet：

|  |
| --- |
| **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  response.setContentType("text/html");  response.setCharacterEncoding("utf-8");  PrintWriter out = response.getWriter();  out  .println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\">");  out.println("<HTML>");  out.println(" <HEAD><TITLE>Hacked by kxlzx</TITLE></HEAD>");  out.println(" <BODY>");  out.print(" 传给我一个jsonObject，返回jsonobject中name的值<br>");  out.print(" 示例：http://www.inbreak.net/HashDos/ TestJsonObject?jsonobject={name:'kxlzx'}<br>");  out.print(" <br>");  out.print(" <br>");  String json = request.getParameter("jsonobject");  System.*out*.println(json);  **try** {  **if** (json != **null**) {  JSONObject jsonObject = **new** JSONObject(json);  out.print("name=" + jsonObject.get("name"));  }  } **catch** (JSONException e) {  e.printStackTrace();  }  out.println(" </BODY>");  out.println("</HTML>");  out.flush();  out.close();  } |

当用户提交 http://www.inbreak.net/HashDos/TestJsonObject?jsonobject={name:'kxlzx'} 就会从中读取用户提交json字符串中的name的值。

邪恶的用户又来了，提交：



其中的post数据太大，大概有1M左右，简单的写一下：

|  |
| --- |
| POST http://www.inbreak.net:8080/HashDos/servlet/TestJsonObject HTTP/1.1  Host: www.inbreak.net:8080  jsonobject={name:0,**这里是生成的攻击数据**} |

我们看看服务器的反应：



注意这个图，tomcat版本是6.0.35，Tomcat是java跑起来的，CPU占用，已经到达99%，tomcat直接卡死。

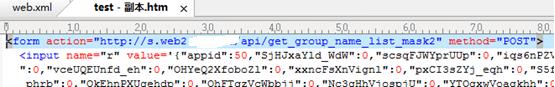
这是服务器上的现象，作为客户端，我们所能看到的，是服务器一直不响应。就像楼下的某web聊天服务器。

**JAVA HASH碰撞拒绝服务 - - 某webim拒绝服务漏洞：**

某web聊天工具，是web2.0的代表，使用ajax传输，长连接保持页面不刷新。

从一个请求的返回的结果中得到，这个应用的背后是个tomcat的集群，必须是java写的web应用。并且使用json传输，虽然不知道是不是jsonObject，但是主流的第三方json解析框架，都使用了hashmap，这样的架构，随便找到一个点都可以攻击。这是一个典型的JSONOBJECT拒绝服务攻击的示例。

为了方便使用，可以直接写个form，随时可以提交。



表单中的“R”的值，是一个JSON字符串，原本内容为

      R= {"appid":50}

我们可以直接在这个基础上构造

      R= {"appid":50,**这里是生成的攻击数据**}

提交表单，等啊等，一直无响应，最后终于看到了。

虽然看不到服务器cpu到了多少，但是从“The server didn't respond in time.”字义上来看，应该是跑去小黑屋面壁了。

**JAVA框架**

其实本文还有一个章节，是DWR的DOS漏洞分析，在当时威力更大，但是xcon2012中已经发布，所以不再详述，如果有兴趣可以去搜索《Xcon2012 攻击JAVA WEB》。

JAVA框架中，还有很多类似的点，不仅仅是在web中，大多数非HTTP传输也都具备JSON接口，有些RMI也都在用hash map做解析，作者没有去找攻击实例，靠这些框架出补丁修补，也不知道会等到什么时候。

**总结**

作者并没有研究python等语言的补丁情况，如果有类似，可能还会存在漏洞。Java漏洞却由tomcat补，从我的安全理念上考虑，这本身就是一件不靠谱的事情，只要深入的研究，肯定是有收获的，也许多搭建几个环境，就能找出没有被公布的东西来。

针对这种攻击的防护，只能先在应用程序中，做一层拦截（WAF），对于java底层的改动，也只能等待java自己出补丁了。由于时间比较久远，这篇文章在2012年写好，一直没有发布，事实上，JAVA在某次更新中，已经修补了此类漏洞，所以大家只有保持最新即可。

**链接**

Hash碰撞导致的denial of service

<http://hi.baidu.com/cn_ben/blog/item/d8690ed1809f0891a1ec9c6f.html>

Xcon2012 攻击JAVA WEB议题下载

<http://www.inbreak.net/archives/477>

阿里巴巴公司欢迎社会各界向我们反馈阿里的安全问题，更多信息可见[报告漏洞](https://security.alibaba.com/leak/report.htm)。