
Tomcat 4.x and Linux

Protecting Tomcat with Linux iptables

Author: Jon Barnett

Document version: 1.0

Table of contents

1	Tomcat 4.x as a Linux production service	1-1
1.1	Tomcat	1-1
1.2	Security	1-1
1.3	Tomcat listener configuration	1-1
1.4	Port redirection.....	1-2
1.4.1	Single machine with one network interface.....	1-2
1.4.2	Single machine with two network interfaces.....	1-3
1.4.3	Two machines in firewall – content server mode	1-5
1.5	Enabling your rules	1-6
1.6	Working with dynamic addresses	1-6

1 Tomcat 4.x as a Linux production service

This guide provides some assistance with the configuration of Tomcat as a production web server on Linux. In order to achieve this, we use the Linux iptables capability to help manage the service. It is assumed that you have some familiarity with building the Linux kernel and that you have created a kernel with iptables enabled in it.

1.1 Tomcat

Tomcat is a fully Java-based servlet container and web server. It is the Sun reference implementation for J2EE servlet container compliance. It provides support for HTTP 1.1 including connection persistence. However, if you want to use it as a production web server for your Linux J2EE installation, there are some protective measures you need to take. This guide assumes that you already have familiarity with Tomcat.

1.2 Security

Generally, you do not want to run any Java-based service as a privileged user. Should there be any security flaws with the Java Virtual Machine (JVM), you do not want your Linux server to be compromised. For example, suppose that there is a problem with the socket implementation that allows a malicious attack to breach Tomcat and the JVM, allowing the attacker to access your Linux box with root privileges.

The problem with using a less privileged account to run the service is that Linux will not allow Tomcat to bind its listeners to ports below 1024 for security reasons. This makes it difficult to operate a web site using the standard port 80 and port 443 for serving content. However, with the use of iptables we can achieve our aims.

1.3 Tomcat listener configuration

The listeners can remain with the default Tomcat settings, binding to port 8080 for serving normal content and port 8443 for serving secure (SSL) content. The proxy ports need to be changed or added so that the apparent content serving ports are inserted into the URLs displayed in the client browser, rather than the actual port to which the listeners are bound. We also want the secure port redirection to operate with port 443, the standard SSL port.

The listener configuration below is for the listener bound to port 8080 but serving content on port 80. Note the proxyPort and the redirectPort settings.

```
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
  port="8080" minProcessors="5" maxProcessors="75"
  enableLookups="true" redirectPort="443"
  acceptCount="10" debug="0" connectionTimeout="20000"
  useURIVValidationHack="false"
  proxyPort="80"/>
```

The listener configuration below is for the listener bound to port 8443 but serving content on port 443. Again, note the proxyPort setting.

```
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
  port="8443" minProcessors="5" maxProcessors="75"
  enableLookups="true"
  acceptCount="10" debug="0" scheme="https" secure="true"
  useURIVValidationHack="false"
  proxyPort="443">
  <Factory className="org.apache.coyote.tomcat4.CoyoteServerSocketFactory"
    keystoreFile="/usr/local/tomcat-4.1/conf/keystore"
    clientAuth="false" protocol="TLS" />
</Connector>
```

You can disable the AJP13 listener as we are not serving content through an Apache web server. Although this is another means of protecting the Tomcat server, it adds complexity and delay to the system.

Starting Tomcat using an account such as the standard apache user defined in most Linux distributions will proceed without any problems. This configuration allows Tomcat to be started with a non-privileged account as the ports to which the listeners are bound are above 1023.

You can now secure the content by disabling login for the account with which you start this service, changing ownership of the content directories and so on.

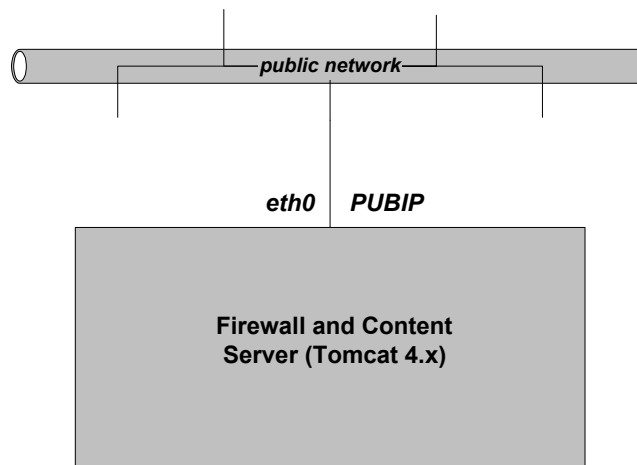
1.4 Port redirection

Tomcat is now configured to operate and deliver content with the correct URL port information in relative links in the content. The redirection service for secure content will also redirect to port 443.

Now we need to configure Linux to redirect requests from port 80 and port 443 to the Tomcat listeners. This requires the use of iptables, a stateful firewall service built into the Linux kernel from version 2.4.x. I will cover three examples based on physical configurations you may have. The assumption in these examples is that the IP tables for the firewall are clear of any other rules and the firewall machine does not otherwise route packets between the networks.

1.4.1 Single machine with one network interface

This configuration consists of a single machine that houses Tomcat. The interface eth0 has an IP address *PUBIP* connected to a public network.



The necessary re-routing is provided by the following commands.

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -d PUBIP -j DNAT --to PUBIP:8080
iptables -t nat -A PREROUTING -p tcp --dport 443 -d PUBIP -j DNAT --to PUBIP:8443
```

The commands route tcp-type connections for the IP address *PUBIP* to the destination port specified by the *dport* option are forwarded to the IP:port combination specified by *to* option. This is accomplished using destination network address translation (DNAT). Since there is no other network interface, there is no need to perform any source network address translation.

An alternative, using the interface definition is:

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -i eth0 -j DNAT --to PUBIP:8080
iptables -t nat -A PREROUTING -p tcp --dport 443 -i eth0 -j DNAT --to PUBIP:8443
```

However, we still need to process for the re-routed destination if the public IP address changes as can happen for dynamic assignments through DHCP or PPP. With this in mind, there is little advantage to using the second method and a few disadvantages.

A local user under these last pair of rules would not be able to access the services in the same manner. The reason for this is that a local user accessing services on *PUBIP:80* or *PUBIP:443* would not activate these rules and therefore the redirection would never occur. Secondly, any access via the external network to a target whose destination port matches port 80 or port 443 would be redirected to *PUBIP:8080* and *PUBIP:8443* respectively. In most instances, this is an undesirable side-effect.

We cannot block access to port 8080 and port 8443 with this configuration as the re-routed packets from port 80 and port 443 respectively are routed to those ports through the same interface specified by the *PUBIP* address.

In theory, you should be able to redirect to 127.0.0.1:8080 and 127.0.0.1:8443 but this does not seem to work, or at least not with kernels up to 2.4.18 on RedHat Linux 7.2 in conjunction with Tomcat 4.x.

Tomcat has a port listener defined for the server itself for management purposes. This is usually defined on port 8005. This is not a port that should be exposed publicly so we need to block packets to the port. This can be achieved by either dropping the packets or explicitly rejecting them. The command is either:

```
iptables -A INPUT -d PUBIP --dport 8005 -j DROP
```

Or:

```
iptables -A INPUT -d PUBIP --dport 8005 -j REJECT
```

Rejecting packets will send return packets to the sender notifying them that their request for a connection has been rejected. This indicates that a service does exist on the port for that IP address. I use the *DROP* method as this does not alert the sender that a service does exist on the port for that address.

You can also drop packets arriving at the port by an interface definition.

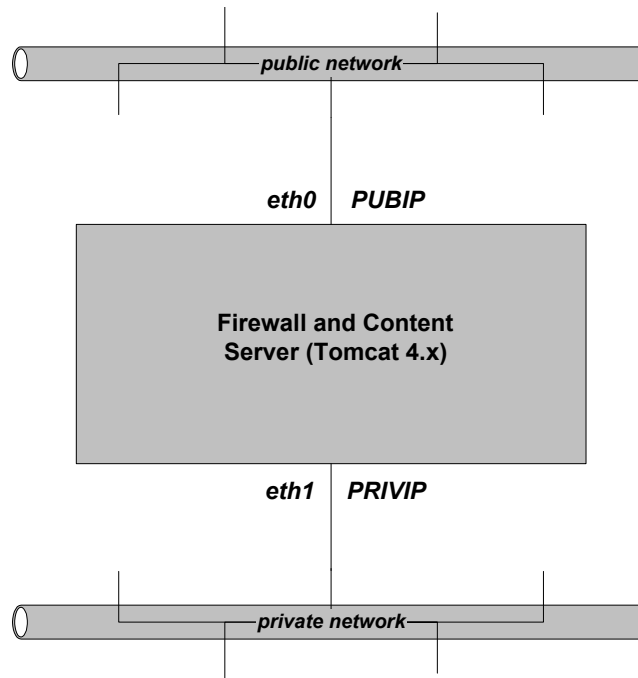
```
iptables -A INPUT -i eth0 --dport 8005 -j REJECT
```

This does not block a local user from accessing the port since the interface is never traversed by the local user whereas the previous command does block the user, based on the IP address target. The local user can access the port via *localhost* in both instances.

No source network address translation (SNAT) is necessary as the machine has the capability of routing packets back to the original requester.

1.4.2 Single machine with two network interfaces

This configuration consists of a single machine that houses Tomcat but this machine has two interfaces. This configuration may occur when the machine also provides a firewall for a private network. In this case, suppose that the interface *eth0* is outward facing (connected the public network) with an IP address of *PUBIP*, and that interface *eth1* is inward facing (connected to the private network) with an IP address of *PRIVIP*. The private network is considered to be a safer zone with respect to malicious attacks.



The necessary re-routing is provided by the following commands.

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -d PUBIP -j DNAT --to PRIVIP:8080
iptables -t nat -A PREROUTING -p tcp --dport 443 -d PUBIP -j DNAT --to PRIVIP:8443
```

This time we redirect the packets to the private network interface represented by the IP address *PRIVIP*. In doing this, we can also hide the actual bound ports on the external interface. We do this by dropping the packets arriving at ports 8080 and 8443. Our commands used for masking ports in use now consist of:

```
iptables -A INPUT -d PUBIP --dport 8005 -j DROP
iptables -A INPUT -d PUBIP --dport 8080 -j DROP
iptables -A INPUT -d PUBIP --dport 8443 -j DROP
```

No SNAT is necessary as the machine has the capability of routing packets back to the original requester.

Using the alternative interface approach can cause problems.

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -i eth0 -j DNAT --to PRIVIP:8080
```

Suppose that internal users (those on the private network) wish to access the service, and their packets are forwarded through the same firewall to reach public IP addresses. The interface rule defined here would never be activated as the packets would pass through interface *eth1* and having reached the target specified by the URL, the packet would not traverse through *eth0*.

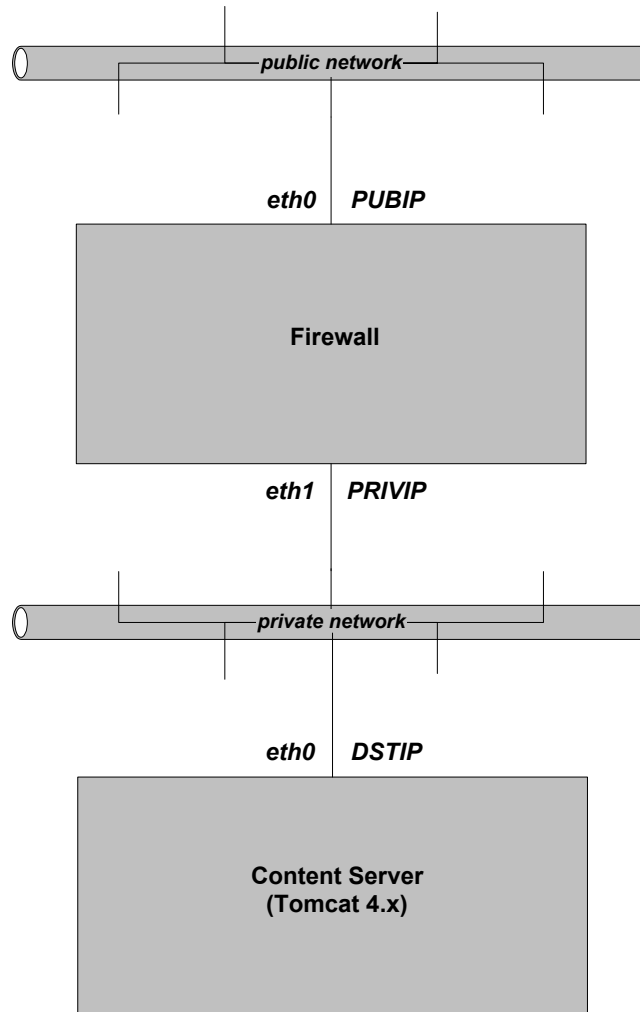
You can counteract this by adding a second rule.

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -i eth1 -j DNAT --to PRIVIP:8080
```

However, this requires that you have symmetrical services on both interfaces – that is, port 80 of both interfaces direct you to the same content. Further, any packet passing through the interface whose target matches the port destination of the rule, will get diverted to *PRIVIP:8080*. As discussed previously, this is usually undesirable.

1.4.3 Two machines in firewall – content server mode

This configuration consists of two machines, the first being a firewall to shield the content server from exposure to the public network and the second being the content server running the Tomcat installation. Suppose in this instance that the interface *eth0* of the firewall is outward facing (connected to the public network) with an IP address of *PUBIP*, and that interface *eth1* of the firewall is inward facing (connected to the private network) with an IP address of *PRIVIP*. The content server has Tomcat configured as previously defined and for the sake of clarity in the example, there are no restrictions on access to Tomcat from the internal network. The target address of the content server is *DSTIP*.



The rules simplify for this case as the firewall has no Tomcat ports to protect locally.

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -d PUBIP -j DNAT --to DSTIP:8080
iptables -t nat -A PREROUTING -p tcp --dport 443 -d PUBIP -j DNAT --to DSTIP:8443
```

We do require SNAT for this configuration. This will modify the source address for the packets routed to the content server so that the sender of the packets appears to be the firewall. This is necessary as the firewall is acting as a proxy to the external request and appears to the requester as the terminating point for the connection. This connection definition will not allow the requester to accept packets that are sent directly from the content server and these are not valid packets for the established connection.

Our SNAT line defines that the request to the content server appears to originate from the private IP address of the firewall.

```
iptables -t nat -A POSTROUTING -o eth1 -j SNAT --to-source PRIVIP
```

That completes the configuration.

There is a side effect of the following commands as discussed previously, particularly if internal users on the private network use the same firewall for outbound request to the public network.

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -i eth0 -j DNAT --to DSTIP:8080
iptables -t nat -A PREROUTING -p tcp --dport 443 -i eth0 -j DNAT --to DSTIP:8443
```

1.5 Enabling your rules

Once your rules are complete to your satisfaction and you have tested them, you can save them and use your boot script to initialise them. RedHat already has a definition for its iptables initialisation, run by an *init.d* script. So with RedHat, you can save the rules in a place where RedHat will use them with the following command.

```
iptables-save > /etc/sysconfig/iptables
```

The only thing you need to do is to enable iptables in the boot scripts. With RedHat, you can use the graphical, systems services configuration interface to disable ipchains and enable iptables. Or you may use the *chkconfig* command line tool to achieve the same.

1.6 Working with dynamic addresses

For one reason or another, you may find yourself working with a network where some of the IP addresses are dynamically assigned. For example, you may be hosting content via a cable connection, or via dynamic IP on ADSL or even via PPP. These environments make it difficult to lock down your iptables rules, particularly if you are using address-based rules.

With some modifications to the saved iptables rules and some modifications to the script to initialise the iptables, you can dynamically reconfigure the rules as addresses change on the firewall.

Suppose that based on the separated firewall that also proxies for internal users, and content server configuration that you have a saved rule set like this:

```
# Generated by iptables-save v1.2.3 on Mon Sep 16 22:26:18 2002
*nat
:PREROUTING ACCEPT [9:1230]
:POSTROUTING ACCEPT [12:890]
:OUTPUT ACCEPT [29:2358]
-A PREROUTING -d 150.101.110.112 -p tcp -m tcp --dport 80 -j DNAT --to-destination
 172.16.20.7:8080
-A PREROUTING -d 150.101.110.112 -p tcp -m tcp --dport 443 -j DNAT --to-destination
 172.16.20.7:8443
-A POSTROUTING -o ppp0 -j SNAT --to-source 150.101.110.112
-A POSTROUTING -o eth1 -j SNAT --to-source 172.16.20.1
COMMIT
# Completed on Mon Sep 16 22:26:18 2002
# Generated by iptables-save v1.2.3 on Mon Sep 16 22:26:18 2002
*filter
:INPUT ACCEPT [1378:105659]
:FORWARD ACCEPT [2:152]
:OUTPUT ACCEPT [1077:111720]
-A FORWARD -i eth1 -j ACCEPT
COMMIT
# Completed on Mon Sep 16 22:26:18 2002
```

The mapping is as follows:

<i>Machine</i>	<i>Interface</i>	<i>IP address</i>	<i>Assignment</i>
Firewall	ppp0	150.101.110.112	Dynamic PPP
	eth1	172.16.20.1	Static
Content server	eth0	172.16.20.7	Static

We want to have at least the dynamic addresses to be represented by variables. The contents are changed to the following:

```
# Generated by iptables-save v1.2.3 on Mon Sep 16 22:26:18 2002
*nat
:PREROUTING ACCEPT [9:1230]
:POSTROUTING ACCEPT [12:890]
:OUTPUT ACCEPT [29:2358]
-A PREROUTING -d PPP0 -p tcp -m tcp --dport 80 -j DNAT --to-destination
  172.16.20.7:8080
-A PREROUTING -d PPP0 -p tcp -m tcp --dport 443 -j DNAT --to-destination
  172.16.20.7:8443
-A POSTROUTING -o ppp0 -j SNAT --to-source PPP0
-A POSTROUTING -o eth1 -j SNAT --to-source ETH1
COMMIT
# Completed on Mon Sep 16 22:26:18 2002
# Generated by iptables-save v1.2.3 on Mon Sep 16 22:26:18 2002
*filter
:INPUT ACCEPT [1378:105659]
:FORWARD ACCEPT [2:152]
:OUTPUT ACCEPT [1077:111720]
-A FORWARD -i eth1 -j ACCEPT
COMMIT
# Completed on Mon Sep 16 22:26:18 2002
```

We now want to modify the existing table initialisation script to pick up the addresses of the interface and substitute them. Using the RedHat `/etc/init.d/iptables` script as an example:

```
start() {
    if [ -f $IPTABLES_CONFIG ]; then
        # Clear pre-existing rules.
        action $"Flushing all current rules and user defined chains:" iptables -F
        action $"Clearing all current rules and user defined chains:" iptables -X
        chains=`cat /proc/net/ip_tables_names 2>/dev/null`
        for i in $chains; do iptables -t $i -F; done && \
            success $"Flushing all current rules and user defined chains:" || \
            failure $"Flushing all current rules and user defined chains:"
        for i in $chains; do iptables -t $i -X; done && \
            success $"Clearing all current rules and user defined chains:" || \
            failure $"Clearing all current rules and user defined chains:"
        for i in $chains; do iptables -t $i -Z; done
        echo $"Determining addresses: "
        PPP0=`ifconfig ppp0 | grep inet | awk -F: '{print $2}' | \
            awk '{print $1}'`
        ETH1=`ifconfig eth1 | grep inet | awk -F: '{print $2}' | \
            awk '{print $1}'`
        if [ "x${PPP0}" = "x" ] ; then
            PPP0="0.0.0.0"
        fi
        if [ "x${ETH1}" = "x" ] ; then
            ETH1=`grep IPADDR \
                /etc/sysconfig/network-scripts/ifcfg-eth1 | \
                awk -F: '{print $2}'`
        fi
        echo $"Applying iptables firewall rules: "
        grep -v "^[[:space:]]*#" $IPTABLES_CONFIG | \
        grep -v "^[[:space:]]*$" | sed -e "s/PPP0/`echo ${PPP0}`/g" | \
        | sed -e "s/ETH1/`echo ${ETH1}`/g" | \
        /sbin/iptables-restore -c && \
        success $"Applying iptables firewall rules" || \
        failure $"Applying iptables firewall rules"
        echo
        touch /var/lock/subsys/iptables
    fi
}
```

When the script runs, it will apply the rules substituting *PPP0* and *ETH1* in */etc/sysconfig/iptables* with the values detected by scanning the interfaces, using *ifconfig*.

The only remaining task is to employ a program or service that monitors changes in the IP address on the firewall external interface and apply the firewall rules when a change does occur. There are programs available, many supplied by dynamic DNS providers.

The only thing to guard against is the order of the scripts when Linux boots. In the default RedHat configuration, the iptables script runs before the network script which can cause problems. The reason is that the modified iptables script invokes the modules for device drivers that support the network interfaces. This can cause the modules to be installed in the wrong order, or devices are bound to the wrong interface. For example, I was binding the device for *eth1* to *eth0* because the iptables script did not *ifconfig eth0* first. You can either change the script execution order with *chkconfig*, or you can *ifconfig eth0* in the iptables script. I prefer changing the execution order even though there is a remote opportunity that you have a network security hole between the time your network interfaces are brought up and your iptables firewall rules are applied.