

IP TABLES

A Beginner's Tutorial

Author : Tony Hill
Date : 24th March 2013
Version : v1-3
Linux OS : Ubuntu 12.04.2 LTS (precise)
Kernel : v3.2.0-39
Iptables : v1.4.12



INDEX

1	INTRODUCTION	2
2	FIREWALL OVERVIEW	2
3	IPTABLES OVERVIEW	3
4	IPTABLES STRUCTURE AND TERMINOLOGY	3
5	BASIC CONFIGURATION	6
5.1	EXAMINING THE TABLES	6
5.2	DEFAULT POLICY.....	6
5.3	INTERFACE RULES	9
5.4	PROTOCOLS & SERVICES – DHCP.....	10
5.5	PROTOCOLS & SERVICES – TCP & UDP.....	11
5.6	BASIC CONFIGURATION SUMMARY.....	13
5.7	BASIC CONFIGURATION ADDITIONAL INFORMATION	13
5.7.1	Skype.....	13
5.7.2	TCP.....	14
5.7.3	FTP.....	16
6	ADVANCED CONFIGURATION	17
6.1	PROTOCOLS & SERVICES – CONNECTION TRACKING	18
6.1.1	Examining the Connections Database.....	18
6.1.2	Conntrack Modules	19
6.1.3	Conntrack Tuning	20
6.1.4	Conntrack Rule Specifications	22
6.2	PROTOCOLS & SERVICES – SSH	22
6.3	PROTOCOLS & SERVICES – FTP	23
6.4	PROTOCOLS & SERVICES – TFTP	24
6.5	PROTOCOLS & SERVICES – DHCP SERVER	25
6.6	PROTOCOLS & SERVICES – DNS SERVER	25
6.7	PROTOCOLS & SERVICES – CIFS/SMB	26
6.7.1	NETBIOS Name Resolution.....	28
6.7.2	NETBIOS Browser Service.....	29
6.7.3	NETBIOS Session Service	29
6.7.4	NETBIOS Filters.....	30
6.8	PROTOCOLS & SERVICES – ICMP	31
6.9	LOGGING	33
6.10	OTHER TABLES – THE MANGLE TABLE.....	36
7	DEPLOYING & AUTO-STARTING THE FIREWALL.....	37
7.1	STEP ONE	37
7.2	STEP TWO.....	40
7.3	STEP THREE.....	40
7.4	STEP FOUR.....	41

1 Introduction

This paper shows how to use iptables to set up a secure firewall on your Linux home computer(s). It contains plenty of configuration examples and command output. If you follow the examples you will be able to build and deploy a robust and flexible firewall of your own.

Having configured the firewall there are instructions on how to create a script to start it automatically at boot-time using the “*/etc/init.d update-rc*” mechanism.

2 Firewall Overview

Essentially, there are two types of firewall - external and internal. Corporate firewalls are usually dedicated external devices with complex rule-sets, whereas internal (personal) firewalls run on your computer and are generally much simpler to configure.

The basic job of both types of firewall is the same – an external firewall prevents unwanted “outside” traffic from entering your network whereas an internal firewall prevents unwanted “inside” traffic from entering your computer; together with any “outside” traffic that the external firewall may have allowed through either deliberately or by misconfiguration.

An underlying principle of all firewalls is as follows:

The outbound traffic that you generate is good because you sent it so you know what it is. Inbound responses to that traffic must, for the most part, also be good. Unsolicited inbound traffic may be bad and should be stopped!

Clearly there are some exceptions to this principle but I am assuming that it holds true for the purpose of this tutorial.

Most corporate firewalls spend their day trying to detect and prevent Denial of Service attacks. They not only enforce connection rules but also look out for anomalous protocol behaviour and use deep packet inspection to find virus signatures and other naughty code. Iptables is very good at the connection rules thing, but is not a virus scanner or deep packet inspection tool.

Note: *Fire-walling and virus protection are two distinct functions. A personal firewall per se does not protect against viruses, although most commercially available personal firewall packages are accompanied by a virus scanner of some description.*

Firewalls correlate related outbound and inbound traffic into “flows”. Related traffic is any bi-directional traffic stream that has corresponding source and destination IP addresses, protocol types, source and destination port numbers and, in the case of TCP, sequence numbers and acknowledgements. The firewall maintains a connection table to track the state of each flow and check for correct protocol behaviour. Firewalls that maintain connection tables are referred to as “stateful” firewalls. Iptables is a stateful firewall.

3 Iptables Overview

Iptables is a suite of powerful directives that hook into the Linux kernel at various stages of the packet processing lifecycle. Figure-1 below shows where iptables sits in relation to the kernel and at which points the hooks into the kernel are provisioned.

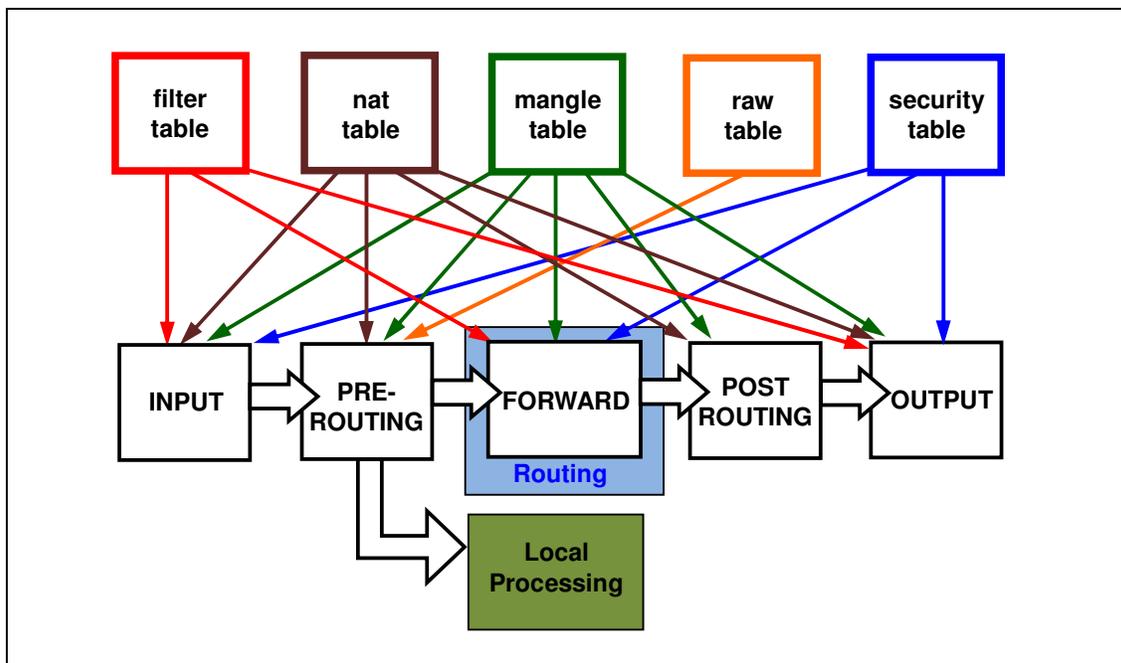


Figure-1: How Iptables Hooks into the Kernel

Iptables is used to create and manage rules that provide, amongst other things, packet manipulation, connection tracking, NAT, ToS / DSCP matching and re-writing, and connection rate-limiting.

Netfilter is the Linux kernel's native packet filtering subsystem which is not available to the user other than through system primitives. Iptables provides a user interface to configure the netfilter subsystem. Most third party Linux firewalls that you download, and install, such as UFW and Firewall Builder, are simply front-ends to iptables. Understanding how to configure iptables natively allows you to implement more granular and comprehensive packet filtering and manipulation policies than any of the third party applications.

4 Iptables Structure and Terminology

Iptables allows an administrator to populate tables with chains of rules that manipulate packets at different stages of the kernel's packet processing life-cycle.

Each table has a distinct function. For example, the **filter table** (the default table) provides commands to filter and accept or drop packets, the **NAT table** provides commands to translate (modify) source or destination IP addresses, and the **mangle table** provides commands to modify packet headers.

Each table contains entities called "chains" under which specific packet rules

(policies) are configured. For example, the *filter table* contains built-in chains called INPUT, FORWARD and OUTPUT. A packet drop rule configured underneath the INPUT chain directs the kernel to DROP packets that are received on a particular interface.

Table-1 below lists the “iptables tables”, their function and the built-in chains they contain. This tutorial focuses predominantly on the *filter table* but the principles apply equally well to all of the tables in the iptables subsystem.

Table	Function	Chain
filter (default)	Packet filtering / firewall	INPUT
		FORWARD
		OUTPUT
NAT	Network Address Translation	PREROUTING
		INPUT
		OUTPUT
		POSTROUTING
mangle	Packet modification	PREROUTING
		INPUT
		FORWARD
		OUTPUT
security	Mandatory Access Control	POSTROUTING
		INPUT
		FORWARD
raw	Bypass “conntrack” for corner cases	OUTPUT
		PREROUTING

Table-1: Iptables Tables & Chains

Figure-2 below shows another representation of the stages at which the various rule chains hook into the kernel packet processing subsystem together with the tables with which the chains are associated. This diagram depicts two types of packet flow:

- Packets entering interfaces one and two that terminate in an application within the computer (local packets). The application returns these packets to the interface over which they arrived if the application issues a response to the sender.
- Packets entering interface one are forwarded direct to interface two, and vice-versa. These packets are not handed over for local processing. The computer is set up to forward packets, which are simply routed through the computer.

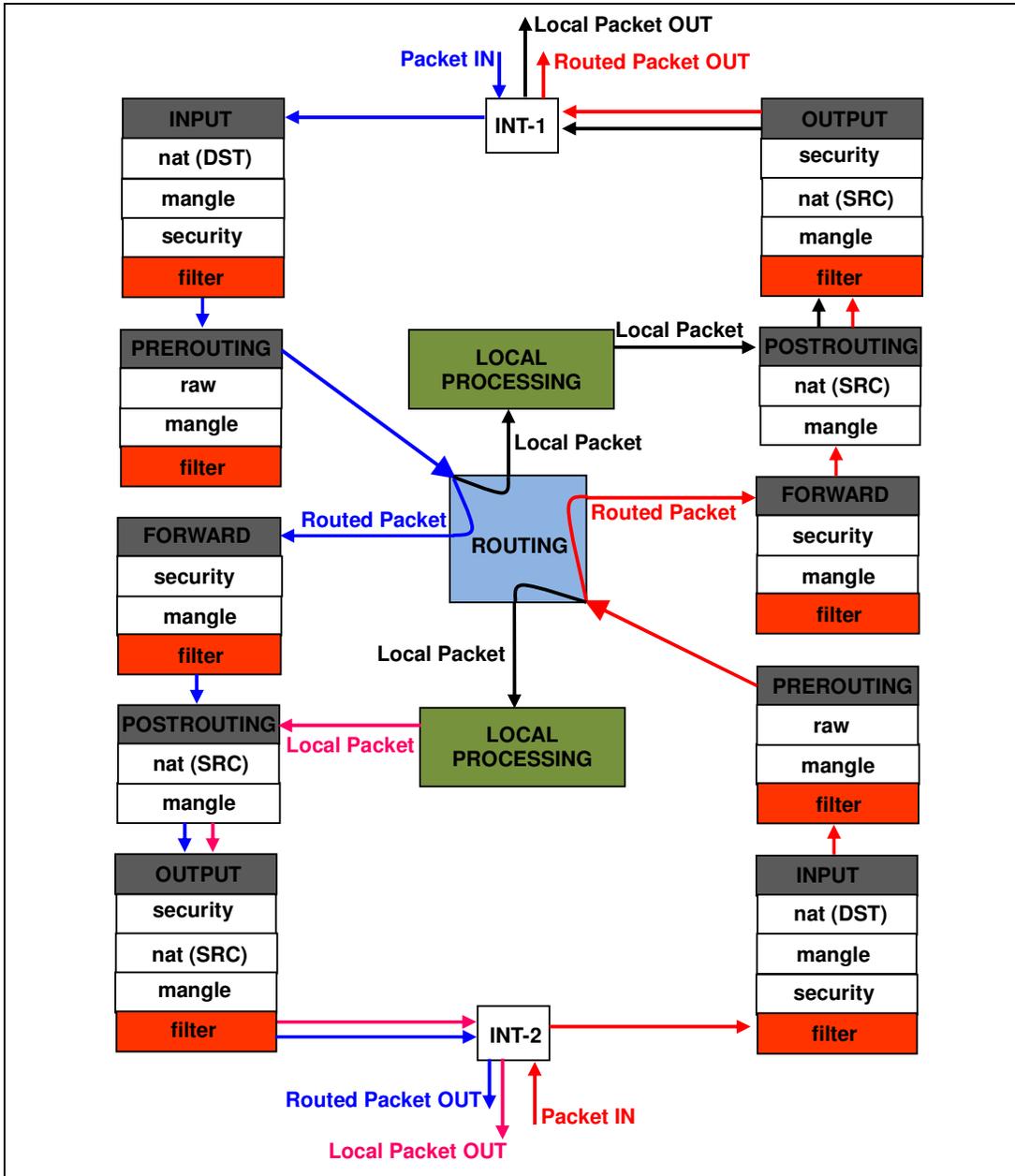


Figure-2: Schematic of Tables & Chains

Iptables makes more sense logically if we visualise the sequence of events from a chains rather than tables perspective. For example, if you configure rules under the INPUT chain within any of the *filter*, *security*, *mangle*, or *nat* tables you apply the actions that those tables support to the packet at the kernel's input processing stage.

Referring to Figure-1 above, if the INPUT chain of the *filter* table contains a rule to “accept” a red packet arriving on INT-2, a rule under the INPUT chain of the *nat* table could be configured to change the packet's destination address and a rule under the INPUT chain of the *mangle* table could be configured to change the packet's Type of Service value.

No rules exist in any of the chains by default and every chain's default policy is ACCEPT. Therefore, by default, iptables allows all packets to pass through the kernel unchanged.

5 Basic Configuration

All of the configuration examples in this section use the *filter table*. If you do not need your computer to perform NAT or to forward (route) packets, the *filter table* is all that you need to build and implement a robust and secure firewall.

In the advanced configuration section I show an example of how the *mangle table* is used to modify the DSCP values of packets before the kernel queues them for transmission on an interface.

5.1 Examining the Tables

To examine the default *filter table* issue the following command, where **-v** is verbose, **-n** is numeric display of addresses and ports, and **-L** is list:

```
root@tony-laptop:~/Firewall# iptables -vnL
Chain INPUT (policy ACCEPT 147 packets, 25720 bytes)
 pkts bytes target    prot opt in     out     source    destination
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target    prot opt in     out     source    destination
Chain OUTPUT (policy ACCEPT 122 packets, 9361 bytes)
 pkts bytes target    prot opt in     out     source    destination
```

The chains within the table are displayed along with their default policy, which is ACCEPT. We have not yet configured any packet rules underneath the chains.

To examine the chains in any of the other tables use the **-t <name>** command, for example:

```
root@tony-laptop:~/Firewall# iptables -t mangle -vnL
```

The *filter table* is the default table so it is not necessary to specify **-t** when examining it.

5.2 Default Policy

Iptables assigns ACCEPT as the default policy to every chain in every table.

Even though iptables isn't yet doing anything special to the packets it is still processing them because the packet and byte counters of each chain are incrementing:

```
root@tony-laptop:~/Firewall# iptables -vnL
Chain INPUT (policy ACCEPT 177 packets, 33983 bytes)
 pkts bytes target    prot opt in     out     source    destination
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target    prot opt in     out     source    destination
```

Tutorial – IPTABLES [Version 1-3]

```
Chain OUTPUT (policy ACCEPT 158 packets, 11814 bytes)
  pkts bytes target    prot opt in     out     source destination
```

Each rule that we configure has its own packet and byte counters, which allows us to check that the rules are processing packets correctly.

To zero the counters on a chain in the default *filter table* specify the **-Z** option together with the name of the chain:

```
iptables -Z INPUT
```

To zero the counters on a chain in any other table specify the table name using the **-t** option:

```
iptables -t mangle -Z OUTPUT
```

The computer is totally “open” if the default policy is ACCEPT on every chain in every table. The first thing to do to start securing the computer is to change the default policy to DROP on the INPUT chain of the *filter table*.

Issue a ping to the loopback IP address 127.0.0.1

```
root@tony-laptop:~/Firewall# ping 127.0.0.1

PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_req=1 ttl=64 time=0.052 ms
64 bytes from 127.0.0.1: icmp_req=2 ttl=64 time=0.039 ms
64 bytes from 127.0.0.1: icmp_req=3 ttl=64 time=0.038 ms
64 bytes from 127.0.0.1: icmp_req=4 ttl=64 time=0.038 ms
```

The ping works as expected. Now change the INPUT chain's default policy to DROP using the **-P** option, and redisplay the chains:

```
iptables -P INPUT DROP

root@tony-laptop:~/Firewall# iptables -nvL

Chain INPUT (policy DROP 0 packets, 0 bytes)
  pkts bytes target    prot opt in     out     source destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target    prot opt in     out     source destination

Chain OUTPUT (policy ACCEPT 29 packets, 1888 bytes)
  pkts bytes target    prot opt in     out     source destination
```

Issue a ping to the loopback interface and examine the *filter table* again:

```
root@tony-laptop:~/Firewall# ping 127.0.0.1

PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
^C
--- 127.0.0.1 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3022ms

root@tony-laptop:~/Firewall# iptables -nvL

Chain INPUT (policy DROP 4 packets, 336 bytes)
```

Tutorial – IPTABLES [Version 1-3]

```
pkts bytes target      prot opt in    out    source      destination
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target      prot opt in    out    source      destination
Chain OUTPUT (policy ACCEPT 4 packets, 336 bytes)
pkts bytes target      prot opt in    out    source      destination
```

The ping fails and the INPUT chain has counted the dropped packets. We sent 4 x 84-byte ICMP packets and the policy counters correctly show 4 x dropped packets and 336 bytes.

This proves that the policy we applied is working. The *filter table* is acting as a firewall and telling the kernel to drop incoming packets that are destined for the loopback interface.

The next step is to change the default policy to DROP on the FORWARD chain, but not on the OUTPUT chain.

```
iptables -P FORWARD DROP
```

We don't change the default policy to DROP on the OUTPUT chain because it is safe to assume that any traffic we send out is secure. We are only concerned with the traffic that we receive and whether any new, unsolicited incoming connections are being attempted. We can always modify the OUTPUT policy and rules later on if we wish to prevent certain types of outbound traffic to specific destinations.

Incidentally, if we do set the OUTPUT chain's policy to drop, we get the following error messages when trying to ping the loopback interface:

```
root@tony-laptop:~/Firewall# ping 127.0.0.1

PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
ping: sendmsg: Operation not permitted
^C
--- 127.0.0.1 ping statistics ---
 4 packets transmitted, 0 received, 100% packet loss, time 3022ms
```

Changing the OUTPUT chain's default policy to DROP instructs the kernel to disallow any outbound packets from every interface on the router, including the loopback interface. An internal ping uses the loopback interface as its source.

I have changed the OUTPUT chain's default policy back to ACCEPT so we can resume sending outbound packets. The *filter table* is displayed below and the packet counters confirm that the kernel is accepting packets for output once more.

```
root@tony-laptop:~/Firewall# iptables -nvL

Chain INPUT (policy DROP 73 packets, 5394 bytes)
pkts bytes target      prot opt in    out    source      destination

Chain FORWARD (policy DROP 0 packets, 0 bytes)
pkts bytes target      prot opt in    out    source      destination

Chain OUTPUT (policy ACCEPT 53 packets, 4155 bytes)
```

pkts	bytes	target	prot	opt	in	out	source	destination
------	-------	--------	------	-----	----	-----	--------	-------------

Notice that the FORWARD chain is not counting any packets. This is because the FORWARD chain is effectively not being used. Why?

On Ubuntu, IP forwarding (a.k.a routing) between interfaces is disabled by default. If you enable IP forwarding the FORWARD chain rules are used only to filter and manipulate packets that the routing subsystem processes.

Please note: On Ubuntu, the command to enable IP forwarding manually is `echo "1" > /proc/sys/net/ipv4/ip_forward`. Use `echo "0" > /proc/sys/net/ipv4/ip_forward` to disable IP forwarding. To enable IP forwarding across reboots uncomment the line `net.ipv4.ip_forward=1` in file `/etc/sysctl.conf`. Be cautious. Enabling IP forwarding on your computer can break your network if you are not sure what you are doing.

5.3 Interface Rules

The default policy in the *filter table's* INPUT chain is currently set to DROP, which is preventing any packets from coming into any interfaces on the computer. We need to append a rule to the INPUT chain to allow packets into the loopback interface because the operating system and some applications need to be able to reach this interface to function properly.

```
iptables -A INPUT -i lo -j ACCEPT
```

The **-A** in the command means “append” the rule to the chain, the **-J** means “jump” to a target, and the target is ACCEPT. In iptables terminology, a “target” is the action to perform on that packet. In the *filter table* the target actions are ACCEPT, DROP, REJECT etc. Different tables support different target actions e.g. the *nat table* supports actions such as SNAT and DNAT to change a packet’s source and/or destination IP addresses.

If you create a user-defined chain you can specify the chain name as a target action in another rule, which allows you to create branching rule-sets. This is explained later on in the section on logging.

Please note: In addition to **-A** the iptables command supports a variety of other arguments, such as **-I** to insert rules, **-D** to delete rules etc. The `iptables -h` command summarises the list of available arguments.

Having added the above rule, examine the *filter table*. Note the use of a new option in the list command to display rule line numbers:

```
root@tony-laptop:~/Firewall# iptables -nvL --line-numbers
```

```
Chain INPUT (policy DROP 55 packets, 6646 bytes)
num  pkts  bytes  target    prot opt in     out     source    destination
 1     33   2268  ACCEPT    all  --  lo      *       0.0.0.0/0  0.0.0/0

Chain FORWARD (policy DROP 0 packets, 0 bytes)
num  pkts  bytes  target    prot opt in     out     source    destination

Chain OUTPUT (policy ACCEPT 97 packets, 6600 bytes)
num  pkts  bytes  target    prot opt in     out     source    destination
```

Rule 1 underneath the INPUT chain directs the kernel to accept into the loopback interface packets from any protocol, from any source (0.0.0.0) to any destination (0.0.0.0) IP address.

Rule 1's packet counter is counting accepted packets but the INPUT chain's summary counter is still counting dropped packets. This is correct because we only applied the ACCEPT action to the loopback interface and the INPUT chain is using its default policy to drop packets arriving on other interfaces.

Ping the loopback interface to check that Rule 1 is working correctly:

```
root@tony-laptop:~/Firewall# ping 127.0.0.1

PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_req=1 ttl=64 time=0.051 ms
64 bytes from 127.0.0.1: icmp_req=2 ttl=64 time=0.052 ms
64 bytes from 127.0.0.1: icmp_req=3 ttl=64 time=0.053 ms
64 bytes from 127.0.0.1: icmp_req=4 ttl=64 time=0.051 ms
^C
--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.051/0.051/0.053/0.008 ms
```

The configuration can now be extended to encompass the “eth0” and “wlan0” interfaces as well as the protocols that run over them.

5.4 Protocols & Services – DHCP

If your computer uses DHCP to obtain IP addresses automatically it is necessary to add an INPUT chain rule to allow its interfaces to receive bootp packets.

DHCP uses the bootp protocol that runs over UDP. The DHCP client uses protocol destination port 68 to receive “bootpc” packets and the DHCP server uses protocol source port 67 to send “bootps” packets (please refer to the IANA protocol port number list).

Create a rule using these port numbers and append it to the INPUT chain, as shown below. Please note that no interfaces are specified in the rule so it will permit “bootp” packets to be received on both the “eth0” and “wlan0” interfaces. But we could have created two separate rules, one for each interface.

```
iptables -A INPUT -p udp --sport 67 --dport 68 -j ACCEPT
```

The INPUT chain's rules are shown below. Notice use of the **-L INPUT** option in the command to display only the rules in the INPUT chain.

```
root@tony-laptop:~/Firewall# iptables -nvL INPUT --line-numbers

Chain INPUT (policy DROP 424 packets, 82256 bytes)
num  pkts  bytes  target    prot  opt  in  out  source      destination
1    1108  94156  ACCEPT    all   --  lo  *    0.0.0.0/0  0.0.0.0/0
2         0      0  ACCEPT    udp   --  *   *    0.0.0.0/0  0.0.0.0/0  udp spt:67 dpt:68
```

Rule 2's counters are zero because the computer already has an IP address so no

“bootp” packets have been exchanged yet with the DHCP server. However, forcing an IP address renewal on “eth0” will cause the counters to increment and, hopefully, the interface should receive an IP address.

```
root@tony-laptop:~/Firewall# dhclient -r
root@tony-laptop:~/Firewall# dhclient eth0
root@tony-laptop:~/Firewall# iptables -nvL INPUT --line-numbers
Chain INPUT (policy DROP 424 packets, 82256 bytes)
num  pkts  bytes  target prot opt in  out  source      destination
 1    1108  94156  ACCEPT  all  --  lo  *    0.0.0.0/0   0.0.0.0/0
 2         2    636  ACCEPT  udp  --  *   *    0.0.0.0/0   0.0.0.0/0  udp spt:67 dpt:68
```

Perfect! The computer receives and counts 2 x “bootp” packets. It sent out a DHCP Discover and the DHCP server returned a DHCP Offer containing our IP, default gateway and DNS addresses. The computer then sent out a DHCP Request to confirm that we will use the offered addresses, and the DHCP server returned a DHCP ACK. We now have all of the IP address information that we need to communicate with the outside world.

Recall that it is not necessary to configure any specific OUTPUT rules because the default output policy is ACCEPT.

5.5 Protocols & Services – TCP & UDP

In order to browse the Internet it is necessary to configure filters that permit inbound responses to the outbound HTTP/HTTPS TCP requests that we transmit. We also need to permit UDP responses for VoIP applications, such as Skype.

The Linux “netstat” command displays the contents of the computer’s TCP connection table. The output below confirms that there are currently no active TCP or UDP connections to any external IP addresses.

```
root@tony-laptop:~/Firewall# netstat -n -A inet
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp    1      0 192.168.1.66:37868      91.189.89.144:80       CLOSE_WAIT
udp    0      0 127.0.0.1:52831        127.0.0.1:53          ESTABLISHED
```

The only connections that do exist are a TCP connection in the CLOSE_WAIT state and an internal ESTABLISHED UDP connection between the computer’s loopback interface and its DNS daemon.

The following commands append rules to the INPUT chain for each of the “eth0” and “wlan0” interfaces to permit reception of inbound responses to outbound connections that we initiate over those interfaces.

Note that the rules do not specify a specific protocol type so any response packets are permitted. The commands include some additional and very important arguments, namely **-m** (match) and **--state ESTABLISHED** and **RELATED**.

```
iptables -A INPUT -i eth0 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Tutorial – IPTABLES [Version 1-3]

```
iptables -A INPUT -i wlan0 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

The **ESTABLISHED** argument permits responses only to connections that we originate. The **RELATED** argument has special significance. It permits responses to connections that existing ESTABLISHED connections may “spawn”. This is explained in more detail in the sections Protocol & Services - FTP and Basic Configuration - FTP.

Please note: The “*nf_conntrack*” kernel module is loaded by default with iptables. It is responsible for identifying established and related IP connections in the connection tracking database. For Passive FTP it is necessary to load the “*nf_conntrack_ftp*” kernel module, which is not loaded by default. This module is specifically responsible for identifying established and related FTP connections.

After appending the new rules to the INPUT chain the *filter table* now looks like this:

```
root@tony-laptop:~/Firewall# iptables -nvL --line-numbers

Chain INPUT (policy DROP 20 packets, 1244 bytes)
num  pkts  bytes  target    prot opt in     out     source    destination
1    187   19204  ACCEPT    all  --  lo     *       0.0.0.0/0  0.0.0.0/0
2     2     636    ACCEPT    udp  --  *      *       0.0.0.0/0  0.0.0.0/0 udp spt:67 dpt:68
3   2698  3070K  ACCEPT    all  --  eth0   *       0.0.0.0/0  0.0.0.0/0 state
RELATED, ESTABLISHED
4     0     0      ACCEPT    all  --  wlan0  *       0.0.0.0/0  0.0.0.0/0 state
RELATED, ESTABLISHED

Chain FORWARD (policy DROP 0 packets, 0 bytes)
num  pkts  bytes  target    prot opt in     out     source    destination

Chain OUTPUT (policy ACCEPT 2511 packets, 315K bytes)
num  pkts  bytes  target    prot opt in     out     source    destination
```

We could have configured more specific rules using the **-p tcp** and **-p udp** arguments but the iptables ESTABLISHED and RELATED options are better because they reduce considerably the number of rules that we need to create.

It is now possible for us to browse the Internet. After browsing to the BBC home page the computer’s netstat connection table shows the external sites to which the computer has established connections successfully. The Linux “netstat” command shows TCP rather than iptables states, the difference between these two is explained in the section Basic Configuration - TCP.

```
root@tony-laptop:~/Firewall# netstat -n -A inet

Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 192.168.1.66:36793     217.41.223.67:80      ESTABLISHED
tcp        0      0 192.168.1.66:51749     217.41.223.66:80      ESTABLISHED
tcp        0      0 192.168.1.66:50812     217.41.223.48:80      ESTABLISHED
tcp        0      0 192.168.1.66:36795     217.41.223.67:80      ESTABLISHED
tcp        0      0 192.168.1.66:36797     217.41.223.67:80      ESTABLISHED
tcp        0      0 192.168.1.66:50813     217.41.223.48:80      ESTABLISHED
tcp        0      0 192.168.1.66:36794     217.41.223.67:80      ESTABLISHED
tcp        1      0 192.168.1.66:37868     91.189.89.144:80      CLOSE_WAIT
tcp        0      0 192.168.1.66:36796     217.41.223.67:80      ESTABLISHED
tcp        0      0 192.168.1.66:34031     212.58.246.94:80      TIME_WAIT
tcp        0      0 192.168.1.66:50814     217.41.223.48:80      ESTABLISHED
tcp        0      0 192.168.1.66:36798     217.41.223.67:80      ESTABLISHED
```

```
tcp      0      0 192.168.1.66:50811    217.41.223.48:80    ESTABLISHED
tcp      0      0 192.168.1.66:33740    173.194.41.99:80    ESTABLISHED
```

The order of the rules is important. If a rule matches a packet the remaining rules take no action. The two rules we just applied are effectively catch-all rules that should be configured at the end of the INPUT chain. If these catch-all rules are configured higher up the chain they may match packets for which more specific rules have been configured lower down.

However, these catch-all rules sometimes work in conjunction with the rules configured above them. For example, when we create the rule to accept inbound FTP connections these catch-all rules are used to process the RELATED connections for Passive FTP to work.

5.6 Basic Configuration Summary

So, what has our pretty basic iptables configuration provided so far?

- By default, the computer drops all inbound packets that are not explicitly ACCEPTED by rules (INPUT default policy is DROP)
- By default, the computer allows out any packets that it generates (OUTPUT default policy is ACCEPT)
- The loopback interface accepts all internally generated packets to ensure correct operation of the operating system and applications
- The “eth0” and “wlan0” interfaces accept DHCP server assigned IP addresses
- The computer is able to browse the web, view videos and participate in audio / video connections with other machines, provided that the computer has initiated the connections
- The computer does not accept any new, inbound connections (but this is fine as we are not yet acting as a web server, DHCP server or any other server for that matter)

For a basic but relatively secure initial configuration that's pretty much it. Type in the aforementioned commands and attempt to access your laptop from another machine. You should find that it is impossible to access anything for which rules have not been explicitly configured.

5.7 Basic Configuration Additional Information

5.7.1 Skype

If the configuration that we just applied doesn't allow new inbound connections, what happens, for example, when someone calls you using Skype? Why does your firewall let this inbound, unsolicited connection into your computer when you have not initiated it?

The answer is, “it isn't unsolicited”.

Skype uses TCP to set up connections and UDP for the transmission of audio and

video data between end-points. The firewall only requires UDP source / destination IP address and source / destination port number information to create flow records.

When you start your Skype client it first establishes a TCP connection with a central Skype server and tells the server the source and destination UDP port numbers that your computer will use for audio sessions. The person calling you will have done exactly the same at his / her end.

The Skype server now has your IP addresses and UDP port information, which it sends to the other party. It also sends the IP address and UDP port information of the other party to your Skype client. The Skype clients at both ends start sending UDP packets direct to each other's IP addresses creating an outbound flow record in their respective firewalls. When the other party calls you, the UDP packets he / she sends contain your UDP destination port and his / her UDP source port information. Your firewall believes that these packets are related to the outbound flow that you just initiated and creates an ESTABLISHED bi-directional flow record allowing the packets in.

5.7.2 TCP

For UDP flows, iptables determines "flow membership" (i.e. bi-directional communication) using only source / destination IP address and source / destination port numbers.

For TCP flows, iptables determines "flow membership" using source / destination IP addresses, source / destination port numbers, and TCP sequence numbers and acknowledgements.

To "talk" TCP, the participating devices must first establish a TCP session. The device initiating the session sends a TCP SYN packet to the receiving device. The receiving device responds with a SYN ACK packet that has the ACK control bit set. The initiating device responds to this SYN ACK packet with an ACK packet that also has the ACK control bit set. This is known as a three-way TCP handshake.

3	0.000249	192.168.1.72	192.168.1.66	TCP	66 49747 > ftp [SYN] Seq=0
4	0.000313	192.168.1.66	192.168.1.72	TCP	66 ftp > 49747 [SYN, ACK]
5	0.000734	192.168.1.72	192.168.1.66	TCP	60 49747 > ftp [ACK] Seq=1

Figure 5-1: Three-Way TCP SYN Handshake

The ACK bit resides in the flags field of the TCP header and is set in every acknowledgement packet for the rest of the conversation thereafter.

```
▼ Flags: 0x010 (ACK)
 000. .... = Reserved: Not set
 ...0 .... = Nonce: Not set
 .... 0... = Congestion Window Reduced (CWR): Not set
 .... .0.. = ECN-Echo: Not set
 .... ..0. = Urgent: Not set
 .... ...1 = Acknowledgement: Set
 .... .... 0... = Push: Not set
 .... .... .0.. = Reset: Not set
 .... .... ..0. = Syn: Not set
 .... .... ...0 = Fin: Not set
```

Figure 5-2: TCP Flags Field

If the ACK bit is set in an inbound packet it indicates that the packet is a response to something we transmitted. Some firewalls and router access control lists use the ACK bit to distinguish whether inbound packets are responses (bit set - therefore permissible) or new connection attempts (bit not set - therefore blocked).

Iptables does not check the ACK bit of TCP packets but it does designate connections as established. The “conntrack” connection tracking subsystem monitors flow records in the iptables connections database. Conntrack shows a connection as being ESTABLISHED after successful completion of the three-way handshake. The output below is an extract from the connections database:

```
[NEW] tcp      6 120 SYN_SENT src=192.168.1.73 dst=192.168.1.66 sport=49511
dport=21 [UNREPLIED] src=192.168.1.66 dst=192.168.1.73 sport=21 dport=49511

[UPDATE] tcp 6 60 SYN_RECV src=192.168.1.73 dst=192.168.1.66 sport=49511
dport=21 src=192.168.1.66 dst=192.168.1.73 sport=21 dport=49511

[UPDATE] tcp 6 432000 ESTABLISHED src=192.168.1.73 dst=192.168.1.66
sport=49511 dport=21 src=192.168.1.66 dst=192.168.1.73 sport=21 dport=49511
[ASSURED]
```

Use of the **ESTABLISHED** argument in a rule effectively tells iptables to check the connections database and directs it to permit inbound packets that are part of an established flow i.e. one that we initiated or permitted to be initiated.

Use of the **RELATED** argument performs has a complementary but slightly different function. Some types of connection are not as straightforward as others. For example, in Passive FTP the client establishes an inbound “control” connection to the server using port 21. The server then generates a random port number and tells the client to use that port to set up a separate, inbound “data” connection to the server. We can’t know in advance what the random port number will be so cannot pre-configure a rule to accommodate the separate inbound connection on that port. The RELATED keyword directs iptables to use the information in its connections database to determine whether the data connection is related to the control connection and, if so, to permit it. Iptables only deems connections to be RELATED if they are associated with pre-existing ESTABLISHED connections.

Please note: It is necessary to load the “nf_conntrack_ftp” kernel module to support Passive FTP. This “helper” module is responsible for identifying that the data connection is related to the already established control connection.

Without the “related” command we would need to configure numerous and much more complex rules.

TCP sessions are closed in an orderly fashion using a four-way handshake, as shown in the figure below.

40	22.466363	192.168.1.66	192.168.1.72	TCP	54 ftp > 49747 [FIN, ACK] Seq=
41	22.466646	192.168.1.72	192.168.1.66	TCP	60 49747 > ftp [ACK] Seq=98 /
42	22.468235	192.168.1.72	192.168.1.66	TCP	60 49747 > ftp [FIN, ACK] Seq=
43	22.468258	192.168.1.66	192.168.1.72	TCP	54 ftp > 49747 [ACK] Seq=283

Figure 5-3: TCP Four-Way FIN Handshake

5.7.3 FTP

There are two types of FTP – Active and Passive. It's good to understand how they work in order to configure the correct rules in your firewall for inbound, client initiated FTP connections. The configuration is explained in the next section.

In both types of FTP, the client connects to the server using a control connection to send commands, such as “get”, “put” and “dir” etc. A separate data connection is used to transfer data. In Active FTP the client initiates the control connection and the server initiates the data connection. In Passive FTP the client initiates both connections.

With an inbound Active FTP connection, the client initiates a TCP control connection to server port 21 and tells the server on what port the client wishes to receive data, usually its initiating source port + 1.

The server then opens a separate data connection from port 20 to the client specified receive port and files are exchanged on this connection. Control traffic and commands continue to be exchanged on the control connection.

This is all fine on a local LAN between two trusted machines. But, if the client uses FTP to retrieve files from a server somewhere on the Internet the client has no real way of knowing whether the server initiated data connection is trustworthy or not.

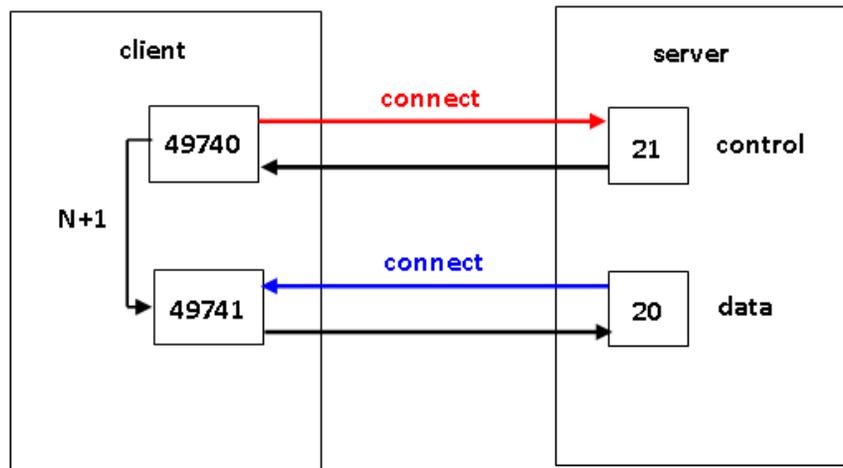


Figure 5-4: Active FTP

Passive FTP was developed to overcome this client side problem. Great! It's now the server firewall administrator's problem. In Passive FTP, the client initiates both the control and data connections.

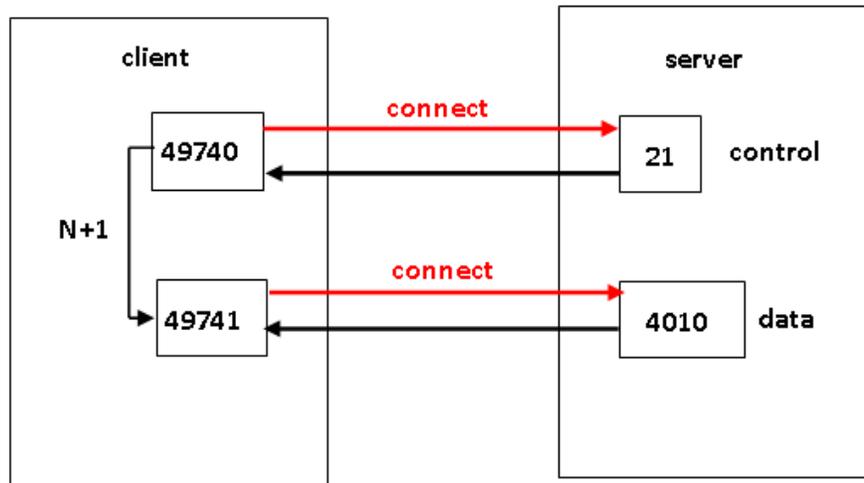


Figure 5-5: Passive FTP

The client connects to the server's control port 21 sending it the "PASV" command indicating that it wishes to use Passive FTP. Meanwhile, the client also opens its own data port - 49741, for example.

If the server agrees to use Passive FTP it generates a random data port number, 4010 for example. The server tells the client on the control connection that it will be listening for a data connection on this port. The client initiates a data connection from its data port 49741 to the server's data port 4010.

In the rules that we configure we must be able to cater for both Active and Passive FTP connections.

For Active FTP, we only need to configure a rule to permit the inbound control port 21 connection from the client. The server's outbound data connection to the client is permitted because we initiate it and our default OUTPUT policy is ACCEPT.

For Passive FTP, we still need to configure a rule to permit the inbound control port connection. But, for the client's inbound data connection we need two things:

1. To specify the ESTABLISHED and RELATED arguments in our general inbound response rule (as configured in the Basic Configuration section)
2. To load the "nf_conntrack_ftp" kernel module.

By the way, the server may well not agree to use Passive FTP, in which case the client and server may agree to fall back to Active FTP.

6 Advanced Configuration

This section provides an overview of connection tracking and also explains how to configure additional rules to expand functionality to allow other connection types into the computer, such as SSH, FTP, TFTP and CIFS (for Samba file sharing).

6.1 Protocols & Services – Connection Tracking

6.1.1 Examining the Connections Database

Iptables maintains state for all the connections it manages using the conntrack subsystem's connections database. Conntrack hooks into the kernel's netfilter APIs. Conntrack creates flow records in its database, tracks and changes the state of existing flows, expires and deletes old flows and calls helper modules, such as `nf_conntrack` and `nf_conntrack_ftp`. These modules are necessary when iptables needs to create flows that are RELATED to ESTABLISHED flows.

The conntrack command line tool is used to inspect the connections database contents interactively.

Please Note: You may have to download and install the conntrack tools separately.

```
apt-get install conntrack conntrack-tools
```

Use the conntrack command with the `-L` option to examine a snapshot of the *filter table's* connections database. You can examine the databases of other tables with the `conntrack <table> -L` command. Use the `-s` or `-d` options to list entries with the specified source or destination IP addresses. The `conntrack -h` command lists all of the other available command options.

The entries below show a TCP dialogue between two computers with IP addresses 192.168.1.66 and 192.168.1.72. The information tells us that this is an Active FTP exchange. The first entry shows that the client (192.168.1.72) has already established a connection to the server's FTP control port 21. The second entry shows that the server has just sent a TCP SYN packet from data port 20 to set up the corresponding FTP data connection to the client.

```
conntrack -L -s 192.168.1.72
tcp      6 431975 ESTABLISHED src=192.168.1.72 dst=192.168.1.66 sport=49747
        dport=21 src=192.168.1.66 dst=192.168.1.72 sport=21 dport=49747
        [ASSURED] mark=0 use=1

Conntrack -L -s 192.168.1.66
tcp      6 119 SYN_SENT src=192.168.1.66 dst=192.168.1.72 sport=20
        dport=49748 src=192.168.1.72 dst=192.168.1.66 sport=49748
        dport=20 [ASSURED] use=2
```

ASSURED means that conntrack will maintain the entries in the database even if the maximum number of sessions has been reached (see the conntrack tuning section).

The first number on the left of each entry is TCP protocol number 6 and the number immediately after it is a count-down timer to flow expiry in seconds. The established connection has 431975 seconds (almost 5 days) until it expires and the SYN sent connection has only 119 seconds left. One or the other of the communicating devices will terminate the established connection well before its countdown timer expires.

You can also examine these records by doing a "cat" of the file:

```
more /proc/net/ip_conntrack
```

In addition to examining the records in the connections database you can also see records in real time as conntrack is creating them. Issue the **conntrack -E** (Events) command:

```
root@tony-laptop:~# conntrack -E

[NEW] tcp      6 120 SYN_SENT src=192.168.1.73 dst=192.168.1.66 sport=49511
dport=21 [UNREPLIED] src=192.168.1.66 dst=192.168.1.73 sport=21 dport=49511
[UPDATE] tcp   6 60 SYN_RECV src=192.168.1.73 dst=192.168.1.66 sport=49511
dport=21 src=192.168.1.66 dst=192.168.1.73 sport=21 dport=49511
[UPDATE] tcp   6 432000 ESTABLISHED src=192.168.1.73 dst=192.168.1.66
sport=49511 dport=21 src=192.168.1.66 dst=192.168.1.73 sport=21 dport=49511
[ASSURED]
[NEW] tcp      6 120 SYN_SENT src=192.168.1.73 dst=192.168.1.66 sport=49512
dport=40253 [UNREPLIED] src=192.168.1.66 dst=192.168.1.73 sport=40253
dport=49512
[UPDATE] tcp   6 60 SYN_RECV src=192.168.1.73 dst=192.168.1.66 sport=49512
dport=40253 src=192.168.1.66 dst=192.168.1.73 sport=40253 dport=49512
[UPDATE] tcp   6 432000 ESTABLISHED src=192.168.1.73 dst=192.168.1.66
sport=49512 dport=40253 src=192.168.1.66 dst=192.168.1.73 sport=40253
dport=49512 [ASSURED]
[UPDATE] tcp   6 120 FIN_WAIT src=192.168.1.73 dst=192.168.1.66 sport=49512
dport=40253 src=192.168.1.66 dst=192.168.1.73 sport=40253 dport=49512
[ASSURED]
[UPDATE] tcp   6 60 CLOSE_WAIT src=192.168.1.73 dst=192.168.1.66 sport=49512
dport=40253 src=192.168.1.66 dst=192.168.1.73 sport=40253 dport=49512
[ASSURED]
[UPDATE] tcp   6 30 LAST_ACK src=192.168.1.73 dst=192.168.1.66 sport=49512
dport=40253 src=192.168.1.66 dst=192.168.1.73 sport=40253 dport=49512
[ASSURED]
[UPDATE] tcp   6 120 TIME_WAIT src=192.168.1.73 dst=192.168.1.66 sport=49512
dport=40253 src=192.168.1.66 dst=192.168.1.73 sport=40253 dport=49512
[ASSURED]
```

6.1.2 Conntrack Modules

Iptables loads most of the modules it needs by default. You can examine the loaded modules using the **lsmod | grep ip** command:

```
root@tony-laptop:~# lsmod | grep ip
ipt_LOG                12783  2
iptable_mangle         12646  0
iptable_nat            13016  0
nf_nat                 24959  1 iptable_nat
nf_conntrack_ipv4     19084  5 iptable_nat,nf_nat
nf_conntrack           73847  4 xt_state,iptable_nat,nf_nat,nf_conntrack_ipv4
nf_defrag_ipv4        12649  1 nf_conntrack_ipv4
iptable_filter         12706  1
ip_tables              18106  3 iptable_mangle,iptable_nat,iptable_filter
x_tables               22011  8
ipt_LOG,xt_limit,xt_state,xt_tcpudp,iptable_mangle,iptable_nat,iptable_filter,ip_tables
```

The modules above include the `nf_conntrack` module, which conntrack needs to create IPv4 and NAT flow records and related connections. However, to support the creation of FTP flow records and related connections it is necessary to load the `nf_conntrack_ftp` module. To load it manually, issue the command:

```
modprobe ip_conntrack_ftp
```

To load the module at boot-time insert the following line in the `/etc/modules` file:

Tutorial – IPTABLES [Version 1-3]

```
root@tony-laptop:~# more /etc/modules
# Load the FTP conntrack module at boot time
nf_conntrack_ftp
```

Confirm that the module is loaded:

```
root@tony-laptop:~/Firewall# lsmod | grep ip (WITH FTP)
ipt_LOG                12783  2
iptable_mangle         12646  0
iptable_nat            13016  0
nf_nat                 24959  1 iptable_nat
nf_conntrack_ipv4     19084  4 iptable_nat,nf_nat
nf_conntrack           73847  6 nf_conntrack_ftp,nf_conntrack_netlink,xt_state,
                        iptable_nat,nf_nat,nf_conntrack_ipv4,nf_defrag_ipv4
12649  1 nf_conntrack_ipv4
iptable_filter         12706  1
ip_tables              18106  3 iptable_mangle,iptable_nat,iptable_filter
x_tables               22011  8
ipt_LOG,xt_limit,xt_state,xt_tcpudp,iptable_mangle,iptable_nat,iptable_filter,ip_tables
```

All of the conntrack modules that are loaded automatically and that are available to be loaded manually are in the following directory:

```
root@tony-laptop:~# ls /lib/modules/3.2.0-38-generic-pae/kernel/net/netfilter/
ipset                nf_conntrack_snmp.ko  xt_conntrack.ko  xt_limit.ko
xt_sctp.ko
ipvs                 nf_conntrack_tftp.ko  xt_cpu.ko        xt_mac.ko
xt_SECMARK.ko       nfnetlink.ko          xt_CT.ko         xt_mark.ko
nf_conntrack_amanda.ko  nfnetlink_log.ko     xt_dccp.ko       xt_multiport.ko
xt_set.ko            nfnetlink_queue.ko   xt_devgroup.ko   xt_NFLOG.ko
nf_conntrack_broadcast.ko  nfnetlink_queue.ko   xt_devgroup.ko   xt_NFLOG.ko
xt_socket.ko        nfnetlink_queue.ko   xt_devgroup.ko   xt_NFLOG.ko
nf_conntrack_ftp.ko  nfnetlink_queue.ko   xt_devgroup.ko   xt_NFLOG.ko
xt_state.ko         nfnetlink_queue.ko   xt_devgroup.ko   xt_NFLOG.ko
nf_conntrack_h323.ko  nf_tproxy_core.ko    xt_dscp.ko       xt_NFQUEUE.ko
xt_statistic.ko     x_tables.ko          xt_DSCP.ko       xt_NOTRACK.ko
nf_conntrack_irc.ko  x_tables.ko          xt_DSCP.ko       xt_NOTRACK.ko
xt_string.ko        xt_addrtype.ko       xt_esp.ko        xt_osf.ko
nf_conntrack.ko     xt_addrtype.ko       xt_esp.ko        xt_osf.ko
xt_tcpmss.ko        xt_AUDIT.ko          xt_hashlimit.ko  xt_owner.ko
nf_conntrack_netbios_ns.ko  xt_AUDIT.ko          xt_hashlimit.ko  xt_owner.ko
xt_TCPMSS.ko        xt_AUDIT.ko          xt_hashlimit.ko  xt_owner.ko
nf_conntrack_netlink.ko  xt_CHECKSUM.ko       xt_helper.ko     xt_physdev.ko
xt_TCPOPTSTRIP.ko    xt_CHECKSUM.ko       xt_helper.ko     xt_physdev.ko
nf_conntrack_pptp.ko  xt_CLASSIFY.ko       xt_hl.ko         xt_pkttype.ko
xt_tcpudp.ko         xt_CLASSIFY.ko       xt_hl.ko         xt_pkttype.ko
nf_conntrack_proto_dccp.ko  xt_cluster.ko        xt_HL.ko         xt_policy.ko
xt_TEE.ko            xt_cluster.ko        xt_HL.ko         xt_policy.ko
nf_conntrack_proto_gre.ko  xt_comment.ko        xt_IDLETIMER.ko  xt_quota.ko
xt_time.ko           xt_comment.ko        xt_IDLETIMER.ko  xt_quota.ko
nf_conntrack_proto_sctp.ko  xt_connbytes.ko      xt_iprange.ko    xt_rateest.ko
xt_TPROXY.ko         xt_connbytes.ko      xt_iprange.ko    xt_rateest.ko
nf_conntrack_proto_udplite.ko  xt_connlimit.ko     xt_ipvs.ko       xt_RATEEST.ko
xt_TRACE.ko          xt_connlimit.ko     xt_ipvs.ko       xt_RATEEST.ko
nf_conntrack_sane.ko  xt_connmark.ko       xt_LED.ko        xt_realm.ko
xt_u32.ko            xt_connmark.ko       xt_LED.ko        xt_realm.ko
nf_conntrack_sip.ko  xt_CONNSECMARK.ko    xt_length.ko     xt_recent.ko
```

Only the default `nf_conntrack` and `nf_conntrack_ftp` modules are required for this tutorial.

6.1.3 Conntrack Tuning

The conntrack subsystem can be tuned to, amongst other things, accommodate more flow records in its connections database. The default number of flow records

supported is usually 65535, depending on how much memory is available in the computer. This is more than enough for home computer firewall implementations but if you were configuring iptables for a firewall in a corporate environment it may be necessary to increase this value.

“**sysctl**” is used to manage kernel runtime parameters. The tuneable parameters are contained in the directory */proc/sys/net/netfilter*.

```
root@tony-laptop:~# ls /proc/sys/net/netfilter
nf_conntrack_acct          nf_conntrack_tcp_timeout_close
nf_conntrack_buckets      nf_conntrack_tcp_timeout_close_wait
nf_conntrack_checksum     nf_conntrack_tcp_timeout_established
nf_conntrack_count        nf_conntrack_tcp_timeout_fin_wait
nf_conntrack_events       nf_conntrack_tcp_timeout_last_ack
nf_conntrack_events_retry_timeout nf_conntrack_tcp_timeout_max_retrans
nf_conntrack_expect_max   nf_conntrack_tcp_timeout_syn_recv
nf_conntrack_generic_timeout nf_conntrack_tcp_timeout_syn_sent
nf_conntrack_icmp_timeout nf_conntrack_tcp_timeout_time_wait
nf_conntrack_log_invalid  nf_conntrack_tcp_timeout_unacknowledged
nf_conntrack_max          nf_conntrack_timestamp
nf_conntrack_tcp_be_liberal nf_conntrack_udp_timeout
nf_conntrack_tcp_loose    nf_conntrack_udp_timeout_stream
nf_conntrack_tcp_max_retrans nf_log
```

For example, issue the following command to check the current maximum number of connections allowed:

```
root@tony-laptop:~# sysctl net.netfilter.nf_conntrack_max
net.netfilter.nf_conntrack_max = 65535
```

To manually increase the number of connections allowed issue the following command:

```
sysctl -w net.netfilter.nf_conntrack_max=131072
```

Or edit the */etc/sysctl.conf* file and enter the following lines to increase the default value across reboots:

```
# Conntrack Max
net.netfilter.nf_conntrack_max=131072
```

The other parameters in the */proc/sys/net/netfilter* directory can also be examined and tuned. For example, to display the current default expiry timer for established TCP connections, issue the following command:

```
root@tony-laptop:~# sysctl net.netfilter.nf_conntrack_tcp_timeout_established
net.netfilter.nf_conntrack_tcp_timeout_established = 432000
```

To display a snapshot of the current number of connections in the database, issue the following command:

```
root@tony-laptop:~# sysctl net.netfilter.nf_conntrack_count
net.netfilter.nf_conntrack_count = 11
```

6.1.4 Contrack Rule Specifications

In addition to using contrack to monitor connections, you can also use contrack syntax in the rules that you specify, for example:

```
iptables -A INPUT -p tcp -m contrack --ctstate ESTABLISHED,RELATED
```

The main difference between using **-m contrack** with **--ctstate** instead of **-m state** with **--state** is that “-m contrack” supports a few more match options than “-m state” but the functionality is essentially the same.

The rules that we specify in this tutorial use the **-m state** and **--state** syntax because this provides sufficient granularity for our requirements.

6.2 Protocols & Services – SSH

If you wish to allow other people to log into your computer using SSH you must complete a few tasks in addition to adding rules to iptables:

- Install an SSH server (in this example OpenSSH v5.9)
- Configure the server's `/etc/ssh/sshd_config` file to allow logins from specific users on specific subnets
- Configure an iptables rule to allow inbound connections on SSH TCP destination port 22

The only item added to the default `/etc/ssh/sshd_config` file in this example is:

```
# Allow only specific users
AllowUsers tony@192.168.1.0/24
```

Now add the iptables rule to the *filter table* to allow inbound connections to the SSH daemon. Notice that this rule uses the parameters **-p tcp**, **--dport 22** and **-s 192.168.1.0/24**. Anyone attempting to connect to the computer from any subnet other than 192.168.1.0/24 is rejected, both by the rule and by the entry in the `sshd_config` file:

```
iptables -A INPUT -i eth0 -p tcp --dport 22 -s 192.168.1.0/24 -j ACCEPT
```

Repeat the rule for interface “wlan0” if you wish to allow logins over the wireless interface.

Connect using PUTTY SSH from another machine and examine the packet counters:

```
root@tony-laptop:~/Firewall# iptables -nvL INPUT --line-numbers

Chain INPUT (policy DROP 129 packets, 23953 bytes)
num  pkts  bytes target  prot opt in      out source          destination
1     983  74126 ACCEPT  all  --  lo      *    0.0.0.0/0       0.0.0.0/0
2       1    318 ACCEPT  udp  --  *      *    0.0.0.0/0       0.0.0.0/0  udp spt:67 dpt:68
3    4403  3787K ACCEPT  all  --  eth0    *    0.0.0.0/0       0.0.0.0/0  state
RELATED, ESTABLISHED
4       0      0 ACCEPT  all  --  wlan0   *    0.0.0.0/0       0.0.0.0/0  state
RELATED, ESTABLISHED
```

```
5      2      104 ACCEPT tcp -- eth0 * 192.168.1.0/24 0.0.0.0/0 tcp dpt:22
```

The connection is successful and the rule 5 packet counters are counting correctly the packets and bytes received.

6.3 Protocols & Services – FTP

If you wish to allow other people to retrieve files from your computer using FTP you must complete a few tasks in addition to adding rules to iptables:

- Install an FTP server (in this example VSFTPD v2.3.5)
- If desired, configure the server's `/etc/vsftpd.conf` file (default in this example)
- Configure iptables rules to allow inbound connections on FTP TCP port 21

Configure the rule for the inbound FTP control port 21 connection. As with the SSH rule, we are only permitting inbound FTP connections from other computers on the 192.168.1.0/24 subnet.

```
iptables -A INPUT -i eth0 -p tcp --dport 21 -s 192.168.1.0/24 -j ACCEPT
```

Please note: This rule will support Passive FTP provided that the `nf_conntrack_ftp` module is loaded. Active and Passive FTP are supported when the above rule is used in conjunction with the following rule that we specified in the section *Basic Configuration section Protocols & Services - TCP & UDP*:

```
iptables -A INPUT -i eth0 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Establish an Active FTP connection to the computer from a remote host and “get” a file. The control and data connections are successful.

Examine the *filter table*:

```
root@tony-laptop:~/Firewall# iptables -nvL INPUT --line-numbers

Chain INPUT (policy DROP 819 packets, 163K bytes)
num  pkts bytes target    prot opt in  out  source      destination
1    1962 166K ACCEPT    all  --  lo   *   0.0.0.0/0   0.0.0.0/0
2      1   318 ACCEPT    udp  --  *    *   0.0.0.0/0   0.0.0.0/0 udp spt:67 dpt:68
3    3157 4231K ACCEPT    all  --  eth0 *   0.0.0.0/0   0.0.0.0/0 state
RELATED, ESTABLISHED
4    3157 4231K ACCEPT    all  --  wlan0 *   0.0.0.0/0   0.0.0.0/0 state
RELATED, ESTABLISHED
5      2    104 ACCEPT    tcp  --  eth0 * 192.168.1.0/24 0.0.0.0/0 tcp dpt:22
6      7    364 ACCEPT    tcp  --  eth0 * 192.168.1.0/24 0.0.0.0/0 tcp dpt:21
```

Notice that the Rule 6 packet count increases for the inbound Port 21 control connection from the client, and so does the Rule 3 packet count. This is because the ESTABLISHED argument in Rule 3 causes iptables to accept inbound packets for the outbound Port 20 data connection even though we have not defined a specific

Port 20 rule.

The message exchange is summarised below showing that the client uses destination port 21 to talk to the server's control port (red), and the server uses source port 20 to send data to, and receive data from, the client (blue):

<u>Client</u>		<u>Server</u>
open control	sport 49747 dport 21 --->	
	<--- login exchange --->	
my data port 49748	sport 49747 dport 21 --->	use port 49748 for data
get file	sport 49747 dport 21 --->	
ACK	<--- dport 49748 sport 20 sport 49748 dport 20 --->	open data connection
ACK	<--- dport 49748 sport 20 sport 49748 dport 20 --->	send file
ACK	<--- dport 49748 sport 20 sport 49748 dport 20 --->	send file
close control	<--- FIN handshake --->	
	<--- FIN handshake --->	close data connection

The rules above are also used for a Passive FTP connection. However, the difference is that the `nf_conntrack_ftp` module uses the `RELATED` argument of Rule 3 to permit the client to establish an inbound data connection on the server specified random port even though we haven't configured a specific rule for that port.

6.4 Protocols & Services – TFTP

If you wish to allow other people to retrieve files from your computer using TFTP you must complete a few tasks in addition to adding rules to iptables:

- Install a TFTP server (in this example TFTP-D-HPA v5.2-1)
- Configure the relevant set up file `/etc/default/tftpd-hpa`
- Configure an iptables rule to allow inbound connections on TFTP UDP destination port 69

Configure a rule for the TFTP protocol UDP port 69, as follows:

```
iptables -A INPUT -i eth0 -p udp --dport 69 -s 192.168.1.0/24 -j ACCEPT
```

As with the FTP rule, TFTP connections are only permitted from other computers on the 192.168.1.0/24 subnet.

Establish a TFTP connection to the computer from another machine and “get” a file. The connection and transfer are successful.

Examine the *filter table*:

Tutorial – IPTABLES [Version 1-3]

```
root@tony-laptop:~/Firewall# iptables -nvL INPUT --line-numbers

Chain INPUT (policy DROP 225 packets, 37425 bytes)
num pkts bytes target    prot opt in     out     source           destination
1      423 43720 ACCEPT    all  --  lo      *      0.0.0.0/0        0.0.0.0/0
2         1   318 ACCEPT    udp  --  *      *      0.0.0.0/0        0.0.0.0/0 udp spt:67 dpt:68
3     4134 3746K ACCEPT    all  --  eth0    *      0.0.0.0/0        0.0.0.0/0 state
RELATED, ESTABLISHED
4         0     0 ACCEPT    all  --  wlan0   *      0.0.0.0/0        0.0.0.0/0 state
RELATED, ESTABLISHED
5         1     52 ACCEPT    tcp  --  eth0    *      192.168.1.0/24   0.0.0.0/0 tcp dpt:22
6         1     52 ACCEPT    tcp  --  eth0    *      192.168.1.0/24   0.0.0.0/0 tcp dpt:21
7         2     118 ACCEPT    udp  --  eth0    *      192.168.1.0/24   0.0.0.0/0 udp dpt:69
```

The connection and file transfer are successful and iptables counts the packets correctly.

6.5 Protocols & Services – DHCP Server

This configuration is necessary only if you wish to use the computer as a DHCP server to allocate IP addresses to DHCP clients. I have not installed and enabled DHCP server software for this example.

Configure a rule for protocol UDP ports 67 (bootps) and 68 (bootpc). Note that the source and destination port numbers are in the reverse order to the DHCP client rule that we created previously:

```
iptables -A INPUT -i eth0 -p udp --sport 68 --dport 67 -s 192.168.1.0/24 -j ACCEPT
```

Examine the *filter table*:

```
root@tony-laptop:~/Firewall# iptables -nvL INPUT --line-numbers

Chain INPUT (policy DROP 0 packets, 0 bytes)
num pkts bytes target    prot opt in     out     source           destination
1      455 46150 ACCEPT    all  --  lo      *      0.0.0.0/0        0.0.0.0/0
2         1   318 ACCEPT    udp  --  *      *      0.0.0.0/0        0.0.0.0/0 udp spt:67 dpt:68
3     4161 3754K ACCEPT    all  --  eth0    *      0.0.0.0/0        0.0.0.0/0 state
RELATED, ESTABLISHED
4         0     0 ACCEPT    all  --  wlan0   *      0.0.0.0/0        0.0.0.0/0 state
RELATED, ESTABLISHED
5         1     52 ACCEPT    tcp  --  eth0    *      192.168.1.0/24   0.0.0.0/0 tcp dpt:22
6         1     52 ACCEPT    tcp  --  eth0    *      192.168.1.0/24   0.0.0.0/0 tcp dpt:21
7         1     49 ACCEPT    udp  --  eth0    *      192.168.1.0/24   0.0.0.0/0 udp dpt:69
8         0     0 ACCEPT    udp  --  eth0    *      192.168.1.0/24   0.0.0.0/0 udp spt:68 dpt:67
```

6.6 Protocols & Services – DNS Server

This configuration is necessary only if you wish to use your computer as a DNS server to service DNS lookups from other computers. I have not installed and enabled DNS server software for this example.

Configure a rule for protocols UDP and TCP port 53. Note that DNS clients may use either TCP or UDP to perform lookups:

```
iptables -A INPUT -p udp --dport 53 -j ACCEPT
```

```
iptables -A INPUT -P tcp --dport 53 -j ACCEPT
```

Examine the *filter table*:

```
root@tony-laptop:~/Firewall# iptables -nvL INPUT --line-numbers

Chain INPUT (policy DROP 2 packets, 80 bytes)
num pkts bytes target prot opt in out source destination
1 774 77384 ACCEPT all -- lo * 0.0.0.0/0 0.0.0.0/0
2 1 318 ACCEPT udp -- * * 0.0.0.0/0 0.0.0.0/0 udp spt:67 dpt:68
3 6122 5292K ACCEPT all -- eth0 * 0.0.0.0/0 0.0.0.0/0 state
RELATED, ESTABLISHED
4 0 0 ACCEPT all -- wlan0 * 0.0.0.0/0 0.0.0.0/0 state
RELATED, ESTABLISHED
5 1 52 ACCEPT tcp -- eth0 * 192.168.1.0/24 0.0.0.0/0 tcp dpt:22
6 1 52 ACCEPT tcp -- eth0 * 192.168.1.0/24 0.0.0.0/0 tcp dpt:21
7 1 49 ACCEPT udp -- eth0 * 192.168.1.0/24 0.0.0.0/0 udp dpt:69
8 0 0 ACCEPT udp -- eth0 * 192.168.1.0/24 0.0.0.0/0 udp spt:68 dpt:67
9 0 0 ACCEPT udp -- * * 0.0.0.0/0 0.0.0.0/0 udp dpt:53
10 0 0 ACCEPT tcp -- * * 0.0.0.0/0 0.0.0.0/0 tcp dpt:53
```

6.7 Protocols & Services – CIFS/SMB

CIFS stands for Common Internet File System and SMB for Server Message Block. CIFS is not a protocol but a term that is used to describe the mechanisms and underlying protocols that network servers use to make their file systems available to clients. However, SMB is a protocol. It is used to transfer data between the clients and servers. CIFS is often associated with Windows networks but Linux servers and clients are increasingly being deployed within these environments. Samba software not only allows Linux servers and clients to integrate with Windows networks but also to offer file system services in their own right.

If you wish to use Samba shares to allow Windows and Linux clients to access folders and files on your Linux computer you must complete a few tasks in addition to adding rules to iptables.

This example uses a Windows Vista client laptop called “catherine-pc” that accesses a shared folder and files on a Ubuntu Samba server called “tony-laptop”. The folder it is sharing (the share name) is called “tony-1”.

- Install a Samba server (in this example Samba 2 v3.6.3-2)
- Add a Samba user to the “*smbpasswd*” file from the command line (in this example the user is “tony”). The Samba user must correspond to an existing Linux user:

```
smbpasswd -a tony
```

- Verify from the command line that the user has been added successfully:

```
pdbedit -L -v
```

- Configure the */etc/samba/smb.conf* file and specify the security method in the [global] section:

```
security = user
```

- Configure the */etc/samba/smb.conf* file and specify the workgroup name in the

[global] section:

workgroup = home

- Configure the `/etc/samba/smb.conf` file and specify ports 139 and 445 in the [global] section:

smb ports = 139 445

- Configure the `/etc/samba/smb.conf` file and specify the server's "server string" and "netbios name" in the [global] section:

server string = tony-laptop

netbios name = tony-laptop

- Configure the `/etc/samba/smb.conf` file and specify the server's interfaces in the [global] section:

interfaces = 127.0.0.0/8 eth0

bind interfaces only = no

- Configure the `/etc/samba/smb.conf` file and specify the name resolution order:

name resolve order = lmhosts bcst

- Configure the `/etc/samba/smb.conf` file and set up the share in the "share definitions" section:

[tony-1]

comment = tony-1

path = /home/tony/Desktop/tony-1

writable = yes

browseable = yes

valid users = tony

- Create a "lmhosts" file in the `/etc/samba` directory of the server that includes the IP address and NETBIOS name of the client computer that will be accessing the share:

192.168.1.73 catherine-pc

- issue the following command to tell the (network message block) "nmbd" daemon where to find the "lmhosts" file:

nmbd -H /etc/samba/lmhosts -D

- Restart the "nmbd" daemon for the above change to take effect:

service nmbd restart

- Restart the Samba daemon for the above changes to take effect:

service smbd restart

- Edit the `C:\Windows\System32\drivers\etc\lmhosts` file on the client computer and enter the IP address and NETBIOS name of the server:

192.168.1.66 tony-pc

- Configure iptables rules to allow inbound connections for NETBIOS Name Service (UDP port 137), NETBIOS Datagram Service (UDP port 138), NETBIOS Session Service (TCP port 139) and Microsoft Directory Service (TCP port 445)

Windows networks are based upon the NETBIOS protocol. File servers and clients use NETBIOS over UDP (unreliable) broadcasts for name resolution and to advertise their capabilities and services. NETBIOS over TCP is used to establish (reliable) connections for SMB data transfers.

By default, most Windows clients are configured to support TCP over NETBIOS. This setting is found in Control Panel → Network Adapters → <Adapter Name> → IPv4 → Properties → Advanced → WINS Tab.

6.7.1 NETBIOS Name Resolution

The network administrator assigns a NETBIOS name (up to 15 characters long) to each device. Each name must be resolved to an IP address for the devices to communicate with each other using NETBIOS over UDP and NETBIOS over TCP.

The NETBIOS Name Service (NBNS) performs the name to IP address resolution function using UDP port 137. NBNS is loosely speaking Windows' equivalent of DNS. Names can be resolved in a number of ways - broadcasts, LOCALHOST file on each machine, Host file on each machine, WINS Server or DNS Server on the LAN segment.

Most home networks are small and use NBNS broadcasts or the LOCALHOST file on each machine for name resolution whereas corporate environments use a Windows Internet Name Service (WINS) or DNS server. The Samba server can be configured to act as a WINS server but in this example I am using the LOCALHOST file on the Windows client and Ubuntu Samba server.

With the broadcast method, when a client wishes to communicate with a server it resolves the IP address of the server by issuing a NETBIOS over UDP port 137 broadcast that contains the NETBIOS name of the server. The server responds with its IP address using a UDP port 137 unicast to the client. The packet exchange below shows the client (catherine-pc 192.168.1.73) issuing a NBNS name query broadcast asking who has name "TONY-LAPTOP" and the server (tony-laptop 192.168.1.66) responds with its IP address.

8	5.212435	192.168.1.73	192.168.1.255	NBNS	92	Name query NB TONY-LAPTOP<20>
9	5.212577	192.168.1.66	192.168.1.73	NBNS	104	Name query response NB 192.168.1.66

Figure 6-1: NETBIOS Name Service Messages

However, in this example we use the LOCALHOST method so the client simply reads its LOCALHOST file to retrieve the IP address of the server, and vice-versa.

6.7.2 NETBIOS Browser Service

Windows machines use the broadcast NETBIOS Browser service on UDP port 138 to advertise on the LAN segment their capabilities, the services they offer and the workgroup to which they belong. The Samba server advertises the fact that it is a file server along with its shares using this mechanism.

All of the machines participate in an election to nominate a Local Master Browser (LMB) and a Backup Master Browser (BMB) for the LAN segment. The LMB and BMB collect and collate the browser service advertisements from all the machines. When you click on network neighbourhood in the file explorer of a Windows PC it interrogates the LMB, which returns a browser list of the workgroups, machines, file servers, print servers etc. that are available on the LAN. You can then click on the resources that you wish to access. This mechanism is more efficient than each machine interrogating every other machine on the network every time someone wants to access a resource.

The packet below is an example of a browser service broadcast from “catherine-pc” informing the LMB and BMB of its presence and the services that it offers as a client.

```
1 0.000000 192.168.1.73 192.168.1.255 BROWSER 243 Host Announcement CATHERINE-PC
```

Figure 6-2: NETBIOS Browser Service

It is possible to configure the Samba server with a high priority to ensure that it wins the browser service election. However, in this example I have left the machines to decide amongst themselves which device should be the LMB.

6.7.3 NETBIOS Session Service

Having resolved name to IP address mappings and exchanged service information it is now possible for clients to connect to servers on the LAN and access their file systems and shares. These connections use the Server Message Block (SMB) protocol that runs over NETBIOS over TCP. TCP is used because it is essential that the connections be reliable. TCP ports 139 and 445 are used for the SMB / NETBIOS / TCP connections.

If a client has NETBIOS over TCP enabled it first establishes a connection with the server using SMB / NETBIOS / TCP port 139. A protocol negotiation then takes place and the connection switches to SMB / NETBIOS / TCP port 445. Why?

It is not necessary to run NETBIOS at all if you do not wish to benefit from the device naming and service advertising functionality that NETBIOS provides. In a non-NETBIOS network, the clients connect to servers using SMB direct over TCP port 445. Disabling NETBIOS altogether eliminates the NETBIOS protocol overhead and reduces broadcast traffic but it also eliminates the network neighbourhood functionality that Windows networking provides. In a non-NETBIOS network, clients must connect direct to shares using an explicit IP address and share name path, or obtain this information from, for example, an Active Directory server.

NETBIOS over TCP is enabled and used in this example so the relevant IPTABLES

Tutorial – IPTABLES [Version 1-3]

filters are configured to allow the server (tony-laptop) to receive inbound packets for NETBIOS Name Service on UDP port 137, Browser Service on UDP port 138, Session Service on TCP port 139 and Session Service on TCP port 445.

To instigate the Windows Vista client's connection to the share on the server I clicked on the network neighbourhood icon in the client's file explorer and double-clicked on the server's icon. The client immediately sent an SMB / NETBIOS / TCP port 139 NETBIOS Session Service (NBSS) request to the server. The server responded that it is accepting connections on this port:

14	5.216130	192.168.1.73	192.168.1.66	NBSS	126	Session request, to TONY-LAPTOP<20> from CATHERINE-PC<00>
16	5.217284	192.168.1.66	192.168.1.73	NBSS	58	Positive session response

Figure 6-3: NETBIOS Session Service Messages

The client then sent a Server Service SMB / NETBIOS / TCP port 139 packet to the server asking it to enumerate the shares that it is offering:

60	12.235429	192.168.1.73	192.168.1.66	SRVSVC	238	NetShareEnumAll request
61	12.235672	192.168.1.66	192.168.1.73	SRVSVC	510	NetShareEnumAll response

Figure 6-4: NETBIOS Server Service Messages

To determine whether the server supports SMB / NETBIOS / TCP port 445 connections, the client sent a "Negotiate Protocol Request" on TCP port 445 to the server. The server responds on TCP port 445 signalling that it does support SMB connections on this port:

102	30.781596	192.168.1.73	192.168.1.66	SMB	202	Negotiate Protocol Request
104	30.782255	192.168.1.66	192.168.1.73	SMB	185	Negotiate Protocol Response

Figure 6-5: SMB Protocol Negotiation

The client requests a connection to the server's shared folder "tony-1" using a "Connect AndX Request" on SMB / NETBIOS / TCP port 445 and the server permits the connection:

143	36.965062	192.168.1.73	192.168.1.66	SMB	150	Tree Connect AndX Request, Path: \\TONY-LAPTOP\TONY-1
144	36.965661	192.168.1.66	192.168.1.73	SMB	120	Tree Connect AndX Response

Figure 6-6: SMB Connection Request

The client asks for a list of the files in the shared folder using a "QUERY_FILE_INFO" request on SMB / NETBIOS / TCP port 445 and the server responds with the information:

151	36.971211	192.168.1.73	192.168.1.66	SMB	130	Trans2 Request, QUERY_FILE_INFO
152	36.971324	192.168.1.66	192.168.1.73	SMB	126	Trans2 Response, FID: 0x245e, Q

Figure 6-7: SMB Query Request

The client is now able to access the files in the server's shared directory using SMB.

6.7.4 NETBIOS Filters

Configure the following iptables rules to permit reception on interface "eth0" the protocols discussed above:

NETBIOS Name Service:

```
iptables -A INPUT -i eth0 -p udp --dport 137 -s 192.168.1.0/24 -j ACCEPT
```

NETBIOS Datagram Service:

```
iptables -A INPUT -i eth0 -p udp --dport 138 -s 192.168.1.0/24 -j ACCEPT
```

NETBIOS Session Service

```
iptables -A INPUT -i eth0 -p tcp --dport 139 -s 192.168.1.0/24 -j ACCEPT
```

Microsoft Directory Service

```
iptables -A INPUT -i eth0 -p tcp --dport 445 -s 192.168.1.0/24 -j ACCEPT
```

Use the client laptop to access a shared folder on the Samba server and examine the *filter table*:

```
root@tony-laptop:~/Firewall# iptables -nvL INPUT --line-numbers

Chain INPUT (policy DROP 26 packets, 3371 bytes)
num  pkts bytes target prot opt in  out  source          destination
<snip>
16    1    78 ACCEPT  udp  --  eth0 *  192.168.1.0/24  0.0.0.0/0 udp dpt:137
17    1   242 ACCEPT  udp  --  eth0 *  192.168.1.0/24  0.0.0.0/0 udp dpt:138
18   30 3951 ACCEPT  tcp  --  eth0 *  192.168.1.0/24  0.0.0.0/0 tcp dpt:139
19   43 5577 ACCEPT  tcp  --  eth0 *  192.168.1.0/24  0.0.0.0/0 tcp dpt:445
20   32 16209 ACCEPT  all  --  eth0 *  0.0.0.0/0       0.0.0.0/0 state
RELATED, ESTABLISHED
21    0    0 ACCEPT  all  --  wlan0 *  0.0.0.0/0       0.0.0.0/0 state
RELATED, ESTABLISHED
```

Access to the share is successful and iptables counts the packets for each of the rules we created.

Notice that the server only saw a single Name Service UDP port 137 broadcast packet from the client because we used the LMHOSTS file for name resolution on the client and the server. During the test, the client only broadcast a single NETBIOS Browser Service UDP port 138 packet to announce its presence and capabilities on the LAN segment. Many more of these advertisements will be counted over time from both the client and the server.

There are a significant number of SMB / NETBIOS / TCP port 139 packets because the Vista client used this port initially to connect to the server. The subsequent negotiation with the server caused a switch to the SMB / NETBIOS / TCP port 445 protocol.

6.8 Protocols & Services – ICMP

Configure ICMP rules if you wish to allow other machines to ping your computer and you want to receive ICMP response message types.

Try to ping the “eth0” interface from another machine before configuring the rules. The ping fails.

Create the following rules in the *filter table*:

```
iptables -A INPUT -p icmp --icmp-type destination-unreachable -j ACCEPT
iptables -A INPUT -p icmp --icmp-type source-quench -j ACCEPT
iptables -A INPUT -p icmp --icmp-type time-exceeded -j ACCEPT
iptables -A INPUT -p icmp --icmp-type parameter-problem -j ACCEPT
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
```

Ping the interface again. The ping succeeds (in this example I sent 4 x ping packets).

Examine the *filter table*:

```
Chain INPUT (policy DROP 9 packets, 2307 bytes)
num  pkts bytes target    prot opt in     out     source         destination
<snip>
11   0      0 ACCEPT    icmp  --  *     *     0.0.0.0/0     0.0.0.0/0     icmptype 3
12   0      0 ACCEPT    icmp  --  *     *     0.0.0.0/0     0.0.0.0/0     icmptype 4
13   0      0 ACCEPT    icmp  --  *     *     0.0.0.0/0     0.0.0.0/0     icmptype 11
14   0      0 ACCEPT    icmp  --  *     *     0.0.0.0/0     0.0.0.0/0     icmptype 12
15   4     240 ACCEPT    icmp  --  *     *     0.0.0.0/0     0.0.0.0/0     icmptype 8
```

Iptables counted the 4 x ping packets and bytes correctly.

Issue a traceroute from the computer to a non-existent IP address so that we receive ICMP diagnostic responses:

```
root@tony-laptop:~/Firewall# traceroute -M icmp 10.0.0.1

traceroute to 10.0.0.1 (10.0.0.1), 64 hops max
 1  192.168.1.254 (192.168.1.254) 0.612ms 0.360ms 0.342ms
 2  217.32.147.2 (217.32.147.2) 6.136ms 5.977ms 5.917ms
 3  217.32.147.46 (217.32.147.46) 7.887ms 7.711ms 7.710ms
 4  213.120.156.74 (213.120.156.74) 9.783ms 9.459ms 9.774ms
 5  217.41.168.207 (217.41.168.207) 9.135ms 10.091ms 9.603ms
 6  217.41.168.109 (217.41.168.109) 9.494ms 9.691ms 9.503ms
 7  * * *
^C
```

Examine the *filter table*. A number of time exceeded ICMP (Type 11) messages are returned and accepted by the rule:

```
0      0 ACCEPT    icmp  --  *     *     0.0.0.0/0     0.0.0.0/0     icmptype 3
0      0 ACCEPT    icmp  --  *     *     0.0.0.0/0     0.0.0.0/0     icmptype 4
18  1488 ACCEPT    icmp  --  *     *     0.0.0.0/0     0.0.0.0/0     icmptype 11
0      0 ACCEPT    icmp  --  *     *     0.0.0.0/0     0.0.0.0/0     icmptype 12
0      0 ACCEPT    icmp  --  *     *     0.0.0.0/0     0.0.0.0/0     icmptype 8
```

Ping a non-existent port on another machine so that we receive a ICMP diagnostic responses:

```
root@tony-laptop:~/Firewall# ping 192.168.1.73 50
PING 50 (0.0.0.50) 56(124) bytes of data.
```

```
^C
--- 50 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4004ms
```

Examine the *filter table*. A number of parameter problem ICMP (Type 12) messages are returned and accepted by the rule:

```
0      0 ACCEPT icmp -- * * 0.0.0.0/0 0.0.0.0/0 icmptype 3
0      0 ACCEPT icmp -- * * 0.0.0.0/0 0.0.0.0/0 icmptype 4
18    1488 ACCEPT icmp -- * * 0.0.0.0/0 0.0.0.0/0 icmptype 11
5     600 ACCEPT icmp -- * * 0.0.0.0/0 0.0.0.0/0 icmptype 12
0      0 ACCEPT icmp -- * * 0.0.0.0/0 0.0.0.0/0 icmptype 8
```

This demonstrates that the rules are working correctly and allowing ICMP response messages into the computer.

6.9 Logging

It is possible to create user-defined chains within any of the tables in order to create branched rules.

This section demonstrates how user-defined chains work and how they are used to set up logging.

Delete the following rule from the INPUT chain:

```
15  4  240  ACCEPT icmp -- * * 0.0.0.0/0 0.0.0.0/0 icmptype 8
```

Re-insert it using the **-I** argument with a new **-j** “jump” action of LOGACCEPT to jump to the new chain that we create:

```
iptables -D INPUT 15
```

```
iptables -I INPUT 15 -p icmp --icmp-type echo-request -j LOGACCEPT
```

Create a user-defined chain called LOGACCEPT in the *filter table*. The **-N** signifies “new”:

```
iptables -N LOGACCEPT
```

Create (Append) two new rules underneath the user-defined LOGACCEPT chain:

```
iptables -A LOGACCEPT -m limit --limit 5/min -j LOG --log-prefix "ICMP Allowed: " -
-log-level 7
```

```
iptables -A LOGACCEPT -j ACCEPT
```

When Rule 15 in the INPUT chain matches an ICMP echo request packet, processing “jumps” to the LOGACCEPT chain. The first rule in the LOGACCEPT chain has a target built-in action of **-j LOG**, which instructs iptables to log the event in syslog. Because the action is LOG rather than ACCEPT or DROP, iptables logs the packet and moves to the next rule, which has a target built-in action of ACCEPT to permit the packet. If the second rule had a target action of DROP instead we would log the packet and drop it.

Examine the *filter table*. Note the new, user-defined LOGACCEPT chain at the foot

of the table:

```
root@tony-laptop:~/Firewall# iptables -nvL
```

```
Chain INPUT (policy DROP 14 packets, 1462 bytes)
  pkts bytes target     prot opt in     out     source           destination
    12   968 ACCEPT     all  --  lo      *       0.0.0.0/0        0.0.0.0/0
     0     0 ACCEPT     udp  --  *       *       0.0.0.0/0        0.0.0.0/0  udp spt:67 dpt:68
     0     0 ACCEPT     udp  --  eth0    *       192.168.1.0/24   0.0.0.0/0  udp dpt:69
     0     0 ACCEPT     udp  --  wlan0   *       192.168.1.0/24   0.0.0.0/0  udp dpt:69
     0     0 ACCEPT     tcp  --  eth0    *       192.168.1.0/24   0.0.0.0/0  tcp dpt:20
     0     0 ACCEPT     tcp  --  eth0    *       192.168.1.0/24   0.0.0.0/0  tcp dpt:21
     0     0 ACCEPT     tcp  --  wlan0   *       192.168.1.0/24   0.0.0.0/0  tcp dpt:20
     0     0 ACCEPT     tcp  --  wlan0   *       192.168.1.0/24   0.0.0.0/0  tcp dpt:21
     0     0 ACCEPT     tcp  --  eth0    *       192.168.1.0/24   0.0.0.0/0  tcp dpt:22
     0     0 ACCEPT     tcp  --  wlan0   *       192.168.1.0/24   0.0.0.0/0  tcp dpt:22
     0     0 ACCEPT     icmp --  *       *       0.0.0.0/0        0.0.0.0/0  icmp type 3
     0     0 ACCEPT     icmp --  *       *       0.0.0.0/0        0.0.0.0/0  icmp type 4
     0     0 ACCEPT     icmp --  *       *       0.0.0.0/0        0.0.0.0/0  icmp type 11
     0     0 ACCEPT     icmp --  *       *       0.0.0.0/0        0.0.0.0/0  icmp type 12
     0     0 LOGACCEPT  icmp --  *       *       0.0.0.0/0        0.0.0.0/0  icmp type 8
     0     0 ACCEPT     udp  --  eth0    *       192.168.1.0/24   0.0.0.0/0  udp dpt:137
     0     0 ACCEPT     udp  --  wlan0   *       192.168.1.0/24   0.0.0.0/0  udp dpt:137
     1   242 ACCEPT     udp  --  eth0    *       192.168.1.0/24   0.0.0.0/0  udp dpt:138
     0     0 ACCEPT     udp  --  wlan0   *       192.168.1.0/24   0.0.0.0/0  udp dpt:138
    30  3284 ACCEPT     tcp  --  eth0    *       192.168.1.0/24   0.0.0.0/0  tcp dpt:139
     0     0 ACCEPT     tcp  --  wlan0   *       192.168.1.0/24   0.0.0.0/0  tcp dpt:139
     0     0 ACCEPT     tcp  --  eth0    *       192.168.1.0/24   0.0.0.0/0  tcp dpt:445
     0     0 ACCEPT     tcp  --  wlan0   *       192.168.1.0/24   0.0.0.0/0  tcp dpt:445
    15  7153 ACCEPT     all  --  eth0    *       0.0.0.0/0        0.0.0.0/0  state
RELATED, ESTABLISHED
     0     0 ACCEPT     all  --  wlan0   *       0.0.0.0/0        0.0.0.0/0  state
RELATED, ESTABLISHED
```

```
Chain FORWARD (policy DROP 0 packets, 0 bytes)
  pkts bytes target     prot opt in     out     source           destination
```

```
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target     prot opt in     out     source           destination
```

```
Chain LOGACCEPT (1 references)
```

```
  pkts bytes target     prot opt in     out     source           destination
     0     0 LOG       all  --  *       *       0.0.0.0/0        0.0.0.0/0
  limit: avg 5/min burst 5 LOG flags 0 level 7 prefix "ICMP Allowed: "

     0     0 ACCEPT     all  --  *       *       0.0.0.0/0        0.0.0.0/0
```

The parameters that we applied to the first rule in the LOGACCEPT chain govern how logging is performed.

The **limit** parameter stipulates that we only log 5 messages per minute with a possible **burst** capability of a further 5 messages. This prevents large volumes of traffic from swamping the logging mechanism causing log files to roll over frequently.

The logging level is **7**, which is syslog level debug. We prefix the log entries with “**ICMP Allowed**” so they are easy to identify in the log file.

Logging is carried out to the `/var/log/syslog` file.

Tutorial – IPTABLES [Version 1-3]

To prove that logging is working, ping the computer from another machine whilst observing the syslog file with the “`tail -f /var/log/syslog`” command. The log messages are generated and displayed correctly:

```
Mar  1 06:15:39 tony-laptop kernel: [32556.569396] ICMP Allowed: IN=eth0
OUT= MAC=00:1d:72:f8:99:80:00:1d:72:d6:ca:d2:08:00 SRC=192.168.1.73
DST=192.168.1.66 LEN=60 TOS=0x00 PREC=0x00 TTL=128 ID=9023 PROTO=ICMP
TYPE=8 CODE=0 ID=1 SEQ=5
```

```
Mar  1 06:15:41 tony-laptop kernel: [32557.579744] ICMP Allowed: IN=eth0
OUT= MAC=00:1d:72:f8:99:80:00:1d:72:d6:ca:d2:08:00 SRC=192.168.1.73
DST=192.168.1.66 LEN=60 TOS=0x00 PREC=0x00 TTL=128 ID=9024 PROTO=ICMP
TYPE=8 CODE=0 ID=1 SEQ=6
```

```
Mar  1 06:15:42 tony-laptop kernel: [32558.593662] ICMP Allowed: IN=eth0
OUT= MAC=00:1d:72:f8:99:80:00:1d:72:d6:ca:d2:08:00 SRC=192.168.1.73
DST=192.168.1.66 LEN=60 TOS=0x00 PREC=0x00 TTL=128 ID=9025 PROTO=ICMP
TYPE=8 CODE=0 ID=1 SEQ=7
```

```
Mar  1 06:15:43 tony-laptop kernel: [32559.607641] ICMP Allowed: IN=eth0
OUT= MAC=00:1d:72:f8:99:80:00:1d:72:d6:ca:d2:08:00 SRC=192.168.1.73
DST=192.168.1.66 LEN=60 TOS=0x00 PREC=0x00 TTL=128 ID=9027 PROTO=ICMP
TYPE=8 CODE=0 ID=1 SEQ=8
```

Examine the *filter table*. Note that the 4 x ICMP packets are counted both in the INPUT chain that instigated the branch to the LOGACCEPT chain, and also in the LOGACCEPT chain itself:

```
root@tony-laptop:~/Firewall# iptables -nvL
```

```
Chain INPUT (policy DROP 245 packets, 36364 bytes)
```

```
pkts bytes target      prot opt in  out  source      destination
```

```
<snip>
```

```
  0      0 ACCEPT      icmp -- *   *   0.0.0.0/0  0.0.0.0/0  icmptype 3
  0      0 ACCEPT      icmp -- *   *   0.0.0.0/0  0.0.0.0/0  icmptype 4
  0      0 ACCEPT      icmp -- *   *   0.0.0.0/0  0.0.0.0/0  icmptype 11
  0      0 ACCEPT      icmp -- *   *   0.0.0.0/0  0.0.0.0/0  icmptype 12
  4    240 LOGACCEPT   icmp -- *   *   0.0.0.0/0  0.0.0.0/0  icmptype 8
```

```
<snip>
```

```
Chain LOGACCEPT (1 references)
```

```
pkts bytes target      prot opt in  out  source      destination
```

```
  4    240 LOG          all  -- *   *   0.0.0.0/0  0.0.0.0/0
limit: avg 5/min burst 5 LOG flags 0 level 7 prefix "ICMP Allowed: "
  4    240 ACCEPT     all  -- *   *   0.0.0.0/0  0.0.0.0/0
```

This functionality is extremely powerful. It is possible to create branches to user-defined chains for a number of rule conditions, not just logging.

In addition to controlling logging rates, the limit functions could be used on other rules to rate limit specific connections. I have not tested this yet but it would certainly be one way to rate limit DoS attacks, for example.

6.10 Other Tables – The Mangle Table

The *mangle table* is used to modify packet headers either on input or just before output.

In this example I am using the *mangle table* to change the DSCP value of ICMP packets that I generate just before they exit the “eth0” interface. I only change the DSCP values of those packets that have a source address of 192.168.1.66. The new DSCP value is 14 decimal.

Create a rule in the *mangle table*. The *mangle table* is used because this table supports packet header modification directives whereas the *filter table* does not. The rule is configured under the POSTROUTING chain but it could also have been configured under the OUTPUT chain.

```
iptables -t mangle -A POSTROUTING -p icmp -s 192.168.1.66 -j DSCP --set-dscp 14
```

Examine the *mangle table*:

```
root@tony-laptop:~/Firewall# iptables -t mangle -nvL

Chain PREROUTING (policy ACCEPT 6 packets, 403 bytes)
  pkts bytes target    prot opt in     out     source         destination
Chain INPUT (policy ACCEPT 6 packets, 403 bytes)
  pkts bytes target    prot opt in     out     source         destination
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target    prot opt in     out     source         destination
Chain OUTPUT (policy ACCEPT 3 packets, 202 bytes)
  pkts bytes target    prot opt in     out     source         destination
Chain POSTROUTING (policy ACCEPT 3 packets, 202 bytes)
  pkts bytes target    prot opt in     out     source         destination
    0      0 DSCP      icmp -- *      *      192.168.1.66  0.0.0.0/0 DSCP set 0x0e
```

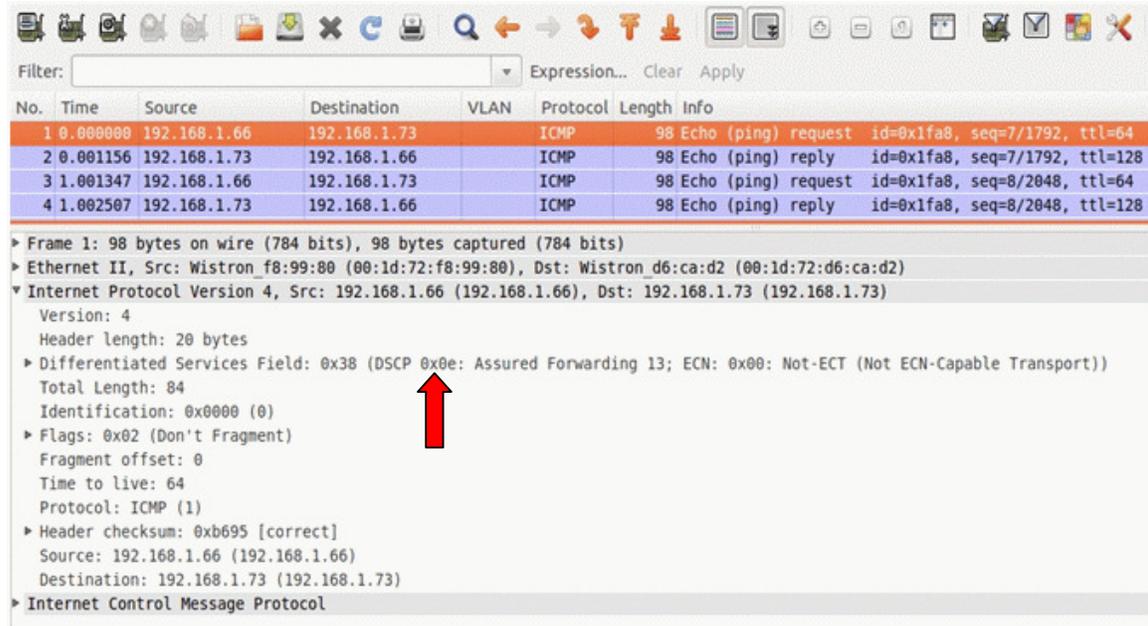
Ping a remote host:

```
root@tony-laptop:~/Firewall# ping 192.168.1.73

PING 192.168.1.73 (192.168.1.73) 56(84) bytes of data.
64 bytes from 192.168.1.73: icmp_req=1 ttl=128 time=1.44 ms
64 bytes from 192.168.1.73: icmp_req=2 ttl=128 time=1.17 ms
64 bytes from 192.168.1.73: icmp_req=3 ttl=128 time=1.15 ms
64 bytes from 192.168.1.73: icmp_req=4 ttl=128 time=1.14 ms
```

Examine the packets using Wireshark. Note the source IP of the packets, which matches the filter. The Differentiated Services Field has been modified to DSCP 0x0e, which is 14 decimal.

Tutorial – IPTABLES [Version 1-3]



The screenshot shows a network traffic capture interface. At the top, there is a toolbar with various icons and a search bar. Below the toolbar is a table of captured packets. The table has columns for No., Time, Source, Destination, VLAN, Protocol, Length, and Info. The first four rows show ICMP ping traffic between 192.168.1.66 and 192.168.1.73. Below the table, there is a detailed view of the selected packet (Frame 1). The details show the Ethernet II header, the Internet Protocol Version 4 header, and the Internet Control Message Protocol header. A red arrow points to the DSCP field in the IP header, which is highlighted in red in the original image. The DSCP field is labeled 'Differentiated Services Field: 0x38 (DSCP 0x0e: Assured Forwarding 13; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))'.

No.	Time	Source	Destination	VLAN	Protocol	Length	Info
1	0.000000	192.168.1.66	192.168.1.73		ICMP	98	Echo (ping) request id=0x1fa8, seq=7/1792, ttl=64
2	0.001156	192.168.1.73	192.168.1.66		ICMP	98	Echo (ping) reply id=0x1fa8, seq=7/1792, ttl=128
3	1.001347	192.168.1.66	192.168.1.73		ICMP	98	Echo (ping) request id=0x1fa8, seq=8/2048, ttl=64
4	1.002507	192.168.1.73	192.168.1.66		ICMP	98	Echo (ping) reply id=0x1fa8, seq=8/2048, ttl=128

Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits)
Ethernet II, Src: Wistron_f8:99:80 (00:1d:72:f8:99:80), Dst: Wistron_d6:ca:d2 (00:1d:72:d6:ca:d2)
Internet Protocol Version 4, Src: 192.168.1.66 (192.168.1.66), Dst: 192.168.1.73 (192.168.1.73)
Version: 4
Header Length: 20 bytes
Differentiated Services Field: 0x38 (DSCP 0x0e: Assured Forwarding 13; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
Total Length: 84
Identification: 0x0000 (0)
Flags: 0x02 (Don't Fragment)
Fragment offset: 0
Time to live: 64
Protocol: ICMP (1)
Header checksum: 0xb695 [correct]
Source: 192.168.1.66 (192.168.1.66)
Destination: 192.168.1.73 (192.168.1.73)
Internet Control Message Protocol

Figure 6-8: DSCP Modification

7 Deploying & Auto-Starting the Firewall

7.1 Step One

Create a script in your home directory called *firewall1.conf*. Use a combination of the commands shown in this tutorial and your own firewall rules changing as required interface names / numbers, IP addresses etc.

The section highlighted in red between the BEGIN INIT INFO and END INIT INFO LABELS is mandatory and must be positioned right after the **#!/bin/sh** line. It must be formatted exactly as shown. Please refer to the file */etc/init.d/skeleton* to see a template file.

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:          firewall1
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Tony's firewall
# Description:       This file should be used to construct scripts to be
#                   placed in /etc/init.d
### END INIT INFO

#
# Author: Tony - 23rd Feb 2013
#

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
NAME=firewall1
DESC="Tony's firewall"
IPT="/sbin/iptables"
```

Tutorial – IPTABLES [Version 1-3]

```
# DO NOT "set -e"

case "$1" in
  start)
    echo "Starting $DESC: $NAME"

# Flush the tables (NAT & MANGLE not currently used but left for reference)
$IPT -F INPUT
$IPT -F OUTPUT
$IPT -F FORWARD
$IPT -F POSTROUTING -t nat
$IPT -F PREROUTING -t nat
$IPT -F PREROUTING -t mangle

# Delete old and create new logging chains
$IPT -X LOGACCEPT
$IPT -X LOGDROP
$IPT -N LOGACCEPT
$IPT -N LOGDROP

# Set default policy to drop everything but allow all OUTPUT
$IPT -P INPUT DROP
$IPT -P OUTPUT ACCEPT
$IPT -P FORWARD DROP

#####
# INPUT section #
#####

# Accept traffic into the loopback interface
$IPT -A INPUT -i lo -j ACCEPT

# Accept services on "eth0" and "wlan0" - DHCP,DNS,TFTP,FTP,SSH,ICMP,CIFS

# DHCP Client (67=bootps, 68=bootpc)
$IPT -A INPUT -p udp --sport 67 --dport 68 -j ACCEPT

# DHCP Server only on eth0 subnet 192.168.1.0/24
# $IPT -A INPUT -i eth0 -p udp --sport 68 --dport 67 -s 192.168.1.0/24 -j ACCEPT

# DNS Server
# $IPT -A INPUT -p udp --dport 53 -j ACCEPT
# $IPT -A INPUT -p tcp --dport 53 -j ACCEPT

# TFTP on specific interfaces on LAN 192.168.1.0/24
$IPT -A INPUT -i eth0 -p udp --dport 69 -s 192.168.1.0/24 -j ACCEPT
$IPT -A INPUT -i wlan0 -p udp --dport 69 -s 192.168.1.0/24 -j ACCEPT

# FTP on specific interfaces on LAN 192.168.1.0/24
# PASSIVE FTP must have nf_conntrack_ftp module loaded in /etc/modules
# Works in conjunction with --state ESTABLISHED, RELATED
$IPT -A INPUT -i eth0 -p tcp --dport 21 -s 192.168.1.0/24 -j ACCEPT
$IPT -A INPUT -i wlan0 -p tcp --dport 21 -s 192.168.1.0/24 -j ACCEPT

# SSH on specific interfaces on LAN 192.168.1.0/24
$IPT -A INPUT -i eth0 -p tcp --dport 22 -s 192.168.1.0/24 -j ACCEPT
$IPT -A INPUT -i wlan0 -p tcp --dport 22 -s 192.168.1.0/24 -j ACCEPT

# ICMP from anywhere (notice the logging)
$IPT -A INPUT -p icmp --icmp-type destination-unreachable -j ACCEPT
$IPT -A INPUT -p icmp --icmp-type source-quench -j ACCEPT
$IPT -A INPUT -p icmp --icmp-type time-exceeded -j ACCEPT
$IPT -A INPUT -p icmp --icmp-type parameter-problem -j ACCEPT
$IPT -A INPUT -p icmp --icmp-type echo-request -j LOGACCEPT

# CIFS - only on specific interfaces on LAN 192.168.1.0/24
# NETBIOS Name Service (name resolution)
$IPT -A INPUT -i eth0 -p udp --dport 137 -s 192.168.1.0/24 -j ACCEPT
```

Tutorial – IPTABLES [Version 1-3]

```
$IPT -A INPUT -i wlan0 -p udp --dport 137 -s 192.168.1.0/24 -j ACCEPT
# NETBIOS Datagram Service (BROWSER service)
$IPT -A INPUT -i eth0 -p udp --dport 138 -s 192.168.1.0/24 -j ACCEPT
$IPT -A INPUT -i wlan0 -p udp --dport 138 -s 192.168.1.0/24 -j ACCEPT
# NETBIOS Session Service (data transfer legacy SMB/NETBIOS/TCP)
$IPT -A INPUT -i eth0 -p tcp --dport 139 -s 192.168.1.0/24 -j ACCEPT
$IPT -A INPUT -i wlan0 -p tcp --dport 139 -s 192.168.1.0/24 -j ACCEPT
# Microsoft Directory Service (data transfer SMB/TCP)
$IPT -A INPUT -i eth0 -p tcp --dport 445 -s 192.168.1.0/24 -j ACCEPT
$IPT -A INPUT -i wlan0 -p tcp --dport 445 -s 192.168.1.0/24 -j ACCEPT

# Accept *all* other safe TCP Established traffic - i.e. traffic that we originate
# Putting this at the end allows all the rules above to count packets
$IPT -A INPUT -i eth0 -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPT -A INPUT -i wlan0 -m state --state ESTABLISHED,RELATED -j ACCEPT

#####
# FORWARD section #
#####
# Possible future use

#####
# OUTPUT section #
#####
# Allow outgoing from local interfaces
# We allow all OUTPUT anyway, but specifying entries shows packet counts per entry
$IPT -A OUTPUT -o lo -j ACCEPT
$IPT -A OUTPUT -o eth0 -j ACCEPT
$IPT -A OUTPUT -o wlan0 -j ACCEPT

#####
# LOGGING Accept #
#####
# User defined chain
# Use -m limit --limit 5/min to log only 5 records per minute to syslog
$IPT -A LOGACCEPT -m limit --limit 5/min -j LOG --log-prefix "ICMP Allowed: " --
log-level 7
$IPT -A LOGACCEPT -j ACCEPT

#####
# LOGGING & Drop #
#####
# User defined chain for future use
$IPT -A LOGDROP -m limit --limit 5/min -j LOG --log-prefix "Disallowed: " --log-
level 7
$IPT -A LOGDROP -j DROP

    echo "Done"
    ;;
stop)
    echo "Disabled $DESC: $NAME - CAUTION! Re-start firewall soonest"
    $IPT -F INPUT
    $IPT -F OUTPUT
    $IPT -F FORWARD
    $IPT -F LOGACCEPT
    $IPT -F LOGDROP
    $IPT -P INPUT ACCEPT
    $IPT -P OUTPUT ACCEPT
    $IPT -P FORWARD ACCEPT
    echo "Done"
    ;;
*)
    N=/etc/init.d/$NAME
    echo "Usage: $N {start|stop}" >&2
    exit 1
    ;;
esac
```

```
exit 0
```

Read through the script carefully. The first thing it does is to flush the chains within the tables and then deletes any existing user-defined chains in the default *filter table*. It recreates the chains and sets the default policy on the INPUT and FORWARD chains to DROP, much as we did earlier on.

All of the policies that I wish to implement are defined within the **case start)** and **case stop)** stanzas. When the firewall is deployed to */etc/init.d*, the script will execute all of the start commands if the argument “start” is passed to the script, and the stop commands if the argument “stop” is passed to the script.

When “stop” is passed to the script, the firewall is effectively turned off i.e. all of the tables are flushed and the default policies changed back to ACCEPT allowing everything in and out.

*Tip: To save a copy of the running firewall at any given moment in time use the **iptables-save > yourfilename** command. To restore this same configuration to the firewall use the **iptables-restore < yourfilename** command.*

7.2 Step Two

Copy the script to the */etc/init.d* directory and call it whatever you want, taking care to not overwrite any existing operating system files that may be called the same name. I called it *firewall1* for this reason.

```
cp $HOME/firewall1.conf /etc/init.d/firewall1
```

Change the permissions of the file as follows:

```
chmod ogu+x /etc/init.d/firewall1
```

7.3 Step Three

Use *update-rc.d* to deploy the script and update the System V “init” script links in the */etc/rc<runlevel>.d* directories. These scripts are run by “init” when it changes run levels and are used to start and stop system services, usually at boot and shutdown time.

Use the following command to update the “init” links:

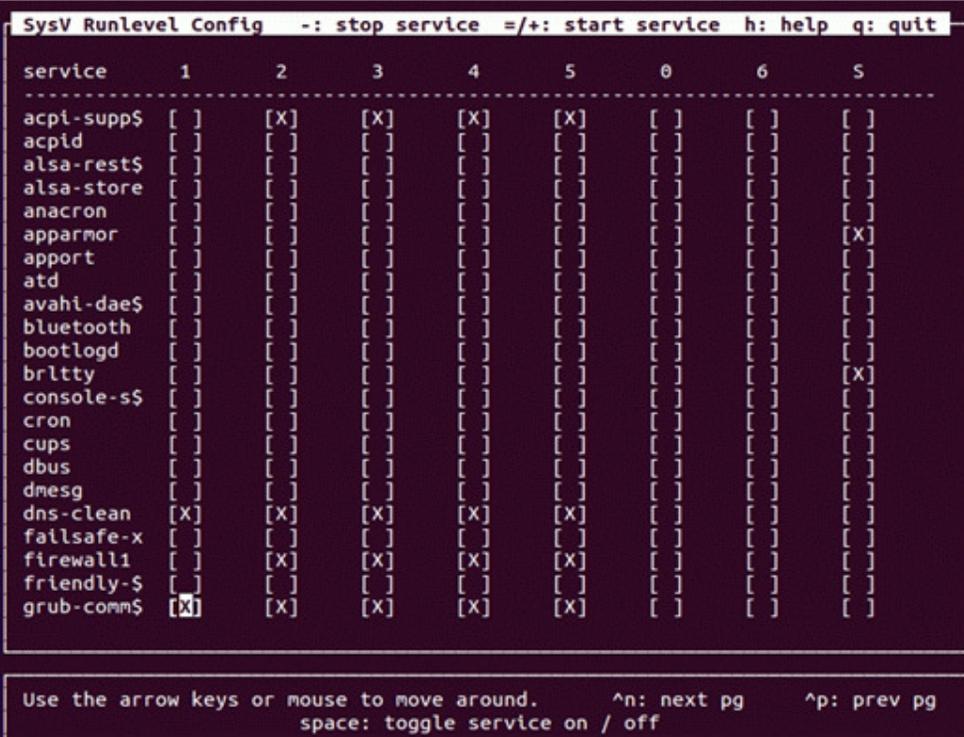
```
update-rc.d firewall1 defaults 20
```

The **defaults** parameter causes the script to be started at run levels 2, 3, 4 and 5, and stopped at run levels 0, 1 and 6. The **20** is the sequence number of the script within those run levels.

If, for whatever reason, you wish to stop the script from loading at boot time, issue the following command:

```
update-rc.d -f firewall1 remove
```

You can apt-get install and use the *sysv-rc-conf* tool to verify that the script's run levels are correct:



```
SysV Runlevel Config  -: stop service  =/+: start service  h: help  q: quit
-----
service    1      2      3      4      5      0      6      S
-----
acpi-supps [ ]    [X]    [X]    [X]    [X]    [ ]    [ ]    [ ]
acpid      [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
alsa-rest$ [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
alsa-store [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
anacron    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
apparmor   [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [X]
apport     [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
atd        [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
avahi-dae$ [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
bluetooth  [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
bootlogd   [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
brltty     [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [X]
console-s$ [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
cron       [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
cups       [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
dbus       [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
dmesg      [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
dns-clean  [X]    [X]    [X]    [X]    [X]    [ ]    [ ]    [ ]
fallsafe-x [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
firewall1  [ ]    [X]    [X]    [X]    [X]    [ ]    [ ]    [ ]
friendly-$ [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]    [ ]
grub-comm$ [X]    [X]    [X]    [X]    [X]    [ ]    [ ]    [ ]
-----
Use the arrow keys or mouse to move around.      ^n: next pg      ^p: prev pg
space: toggle service on / off
```

Figure 7-1: SysV Run Levels

Once you've deployed the firewall, reboot the computer to double check that it has loaded correctly and that all your policies are in place.

7.4 Step Four

Create a small bash script in your home directory that allows you to start and stop the firewall easily when you are logged in.

If you edit the *firewall1.conf* file in your home directory to modify any of the rules, running the bash script with the start argument will copy the file to the */etc/init.d* directory and cause the new policies to be applied immediately.

You **do not** need to repeat the update-rc.d steps.

A sample script is shown below. Remember to change its permissions to execute:

```
chmod u+x firewall.sh
```

Execute the script using either "start" or "stop" as arguments:

```
./firewall.sh start
```

```
./firewall.sh stop
```

```
cat firewall.sh
#!/bin/bash
#
# Script that stops or redeploys and starts the /etc/init.d script
#

# Complain if no parameters passed

if [ "$1" != "start" ] && [ "$1" != "stop" ]; then
    echo "Start/Stop action not supplied!"
    exit
fi

action=$1

if [ "$action" = "stop" ]; then
    /etc/init.d/firewall1 $action
    if [ $? != 0 ]; then echo "Firewall STOP failed"; exit; fi
    iptables -nvL
    echo -e "\nFirewall STOPPED successfully\n"
else
    cp /home/tony/Firewall/firewall1.conf /etc/init.d/firewall1
    if [ $? != 0 ]; then echo "File copy to /etc/init.d failed"; exit; fi
    chmod o-wx /etc/init.d/firewall1
    if [ $? != 0 ]; then echo "User Other permission change failed"; exit; fi
    chmod ug+rx /etc/init.d/firewall1
    if [ $? != 0 ]; then echo "User/Group permission change failed"; exit; fi
    /etc/init.d/firewall1 $action
    if [ $? != 0 ]; then echo "Firewall START failed"; exit; fi
    iptables -nvL
    echo -e "\nFirewall STARTED successfully\n"
fi
```

This completes creation and deployment of the firewall and is the end of this tutorial.