Iptables Tutorial 1.1.19

Oskar Andreasson

Â Â Â Â Â >blueflux@koffein.net>
Â Â Â Â

Copyright Â© 2001-2003 Oskar Andreasson

Dedications

First of all I would like to dedicate this document to mywonderful girlfriend Ninel. She has supported me more than I ever cansupport her to any degree. I wish I could make you just as happy as youmake me.

Second of all, I would like to dedicate this work to all of theincredibly hard working Linux developers and maintainers. It is peoplelike those who make this wonderful operating system possible.

Table of Contents

List of Tables

About the author

 I am someone with too many old computers on his hands. I have myown LAN and want all my machines to be connectedto the Internet, whilst at the same time making myLAN fairly secure. The new iptables is a goodupgrade from the old ipchains in this regard. With ipchains, you couldmake a fairly secure network by dropping all incoming packages notdestined for given ports. However, things like passiveFTP or outgoing DCC inIRC would cause problems. They assign ports onthe server, tell the client about it, and then let the client connect.There were some teething problems in the iptables code that I ran into inthe beginning, and in some respects I found the code not quite ready forrelease in full production. Today, I'd recommend everyone who usesipchains or even older ipfwadm etc .,to upgrade - unless they are happywith what their current code is capable of and if it does what they need.

How to read

This document was written purely so people can start to grasp thewonderful world of iptables. It was never meant to contain information onspecific security bugs in iptables or Netfilter. If you find peculiarbugs or behaviors in iptables or any of the subcomponents, you shouldcontact the Netfilter mailing lists and tell them about the problem andthey can tell you if this is a real bug or if it has already been fixed.There are very rarely actual security related bugs found in iptables orNetfilter, however, one or two do slip by once in a while. These areproperly shown on the front page of the Netfilter main page, and that iswhere you should go to get information on such topics.

The above also implies that the rule-sets available with this tutorial arenot written to deal with actual bugs inside Netfilter. The main goal ofthem is to simply show how to set up rules in a nice simple fashion thatdeals with all problems we may run into. For example, this tutorial willnot cover how we would close down the HTTP port for the simple reason thatApache happens to be vulnerable in version 1.2.12 (This is covered really,though not for that reason).

This document was simply written to give everyone a good and simple primerat how to get started with iptables, but at the same time it was createdto be as complete as possible. It does not contain any targets or matchesthat are in patch-o-matic for the simple reason that it would require toomuch effort to keep such a list updated. If you need information aboutthe patch-o-matic updates, you should read the info that comes with it inpatch-o-matic as well as the other documentations available on the Netfilter main page.

Prerequisites

This document requires some previous knowledge about Linux/Unix, shellscripting, as well as how to compile your own kernel, and some simpleknowledge about the kernel internals.

I have tried as much as possible to eradicate all prerequisites needed beforefully grasping this document, but to some extent it is simply impossible to notneed some previous knowledge.

Conventions used in this document

The following conventions are used in this document when it comes tocommands, files and other specific information.

* Code excerpts and command-outputs are printed like this, with all outputin fixed width font and user-written commands in bold typeface:

```
[blueflux@work1 neigh]$ ls
default  eth0  lo
[blueflux@work1 neigh]$
```

* All commands and program names in the tutorial are shown in boldtypeface.
* All system items such as hardware, and also kernel internals or abstractsystem items such as the loopback interface are all shown in anitalic typeface.
* computer output is formatted in this wayin the text.
* filenames and paths in the file-system are shown like /usr/local/bin/iptables.

ChapterÂ 1.Â Introduction

Table of Contents
Why this document was writtenHow it was writtenTerms used in this document

Why this document was written

 Well, I found a big empty space in the HOWTO's out there lackingin information about the iptables and Netfilter functions in the new Linux2.4.x kernels. Among other things, I'm going to try to answer questionsthat some might have about the new possibilities like state matching. Most of this will be illustrated with an example rc.firewall.txt file that you can use in your /etc/rc.d/ scripts. Yes, this file was originally based upon the masquerading HOWTO for those of you who recognize it.

 Also, there's a small script that I wrote just in case you screwup as much as I did during the configuration available as rc.flush-iptables.txt.

How it was written

I've consulted Marc Boucher and others from the core Netfilter team. Many heartfelt thanks to them for their work and for their help on this tutorial, that I originally wrote for boingworld.com, and now maintain for my own site frozentux.net. This document will guide you through the setup process step by step and hopefully help you to understand some more about the iptables package. I will base most of the stuff here on the example rc.firewall file, since I findthat example a good way to learn how to use iptables. I have decidedto just follow the basic chains and from there go down into each and oneof the chains traversed in each due order. That way the tutorial is a little bit harder to follow, though this way is more logical. Whenever you find something that's hard to understand, just come back to this tutorial.

Terms used in this document

This document contains a few terms that may need more detailed explanations before you read them. This section will try to cover the most obvious ones and how I have chosen to use them within this document.

DNAT - Destination Network Address Translation. DNAT refers to the technique of translating the Destination IP address of a packet, or to change it simply put. This is used together with SNAT to allow several hosts to share a single Internet routable IP address, and to still provide Server Services. This is normally done by assigning different ports with a Internet routable IP address,and then tell the Linux router where to send the traffic.

Stream - This term refers to a connection that sends and receives packets that are related to each other in some fashion. Basically, I have used this term for any kind of connection that sends two or more packets in both directions. In TCP this may mean a connection that sends a SYN and then replies with an SYN/ACK, but it may also mean a connection that sends a SYN and then replies with an ICMP Host unreachable. In other words, I use this term very loosely.

SNAT - Source Network Address Translation. This refers to the techniques used to translate one source address to another in a packet. This is used to make it possible for several hosts to share a single Internet routable IP address, since there is currently a shortage of available IP addresses in IPv4 (IPv6 will solve this).

State - This term refers to which state the packet is in, eitheraccording to RFC 793 - Transmission Control Protocol or according to userside states used in Netfilter/iptables. Note that the used states internally, and externally, do not fully follow the RFC 793 specification fully. The main reason is that Netfilter has to make several assumptions about the connections and packets.

User space - With this term I mean everything and anything that takes placeoutside the kernel. For example, invoking iptables -h takesplace outside the kernel, while iptables -A FORWARD -p tcp -jACCEPT takes place (partially) within the kernel, since a new ruleis added to the ruleset.

Kernel space - This is more or less the opposite of User space. This impliesthe actions that take place within the kernel, and not outside of the kernel.

Userland - See User space.

Chapter 2. Preparations

Table of Contents
Where to get iptablesKernel setupUser-land setupCompiling the user-land applicationsInstallation on Red Hat 7.1

This chapter is aimed at getting you started and to help you understandthe role Netfilter and iptables play in Linux today.This chapter should hopefully get you set up and finished to go with yourexperimentation, and installation of your firewall. Given time andperseverance, you'll then get it to perform exactly as you want it to.

Where to get iptables

The iptables user-space package can bedownloaded from the http://www.netfilter.org/documentation/index.html#FAQ- The official Netfilter Frequently AskedQuestions. Also a good place to start at when wondering whatiptables and Netfilter is about.   . The iptables package also makes use of kernel space facilities which can be configured into the kernel during make configure. The necessary steps will be discussed a bit further down in this document.

Kernel setup

To run the pure basics of iptables you need to configure the following options into the kernel while doing make config or one of its related commands:

CONFIG_PACKET - This option allowsapplications and utilities that needs to work directly to various network devices. Examples of such utilities are tcpdump or snort.

Note

CONFIG_PACKET is strictly speaking not needed for iptables to work, but since it contains so many uses, I have chosen to include it here. If you do not want it, don't include it.

CONFIG_NETFILTER - This option is required if you're going to use your computer as a firewall or gateway to the Internet. In other words, this is most definitely required for anything in this tutorial to work at all. I assume you will want this, since you are reading this.

And of course you need to add the proper drivers for yourinterfaces to work properly, i.e. Ethernet adapter,PPP and SLIP interfaces.The above will only add some of the pure basics in iptables. You won't beable to do anything productive to be honest, it just adds the framework tothe kernel. If you want to use the more advanced options in Iptables, youneed to set up the proper configuration options in your kernel. Here wewill show you the options available in a basic 2.4.9 kernel and a briefexplanation :

CONFIG_IP_NF_CONNTRACK - Thismodule is needed to make connection tracking. Connection tracking is usedby, among other things, NAT andMasquerading. If you need to firewall machines ona LAN you most definitely should mark thisoption. For example, this module is required by the rc.firewall.txt script towork.

CONFIG_IP_NF_FTP - This moduleis required if you want to do connection tracking onFTP connections. SinceFTP connections are quite hard to do connectiontracking on in normal cases, conntrack needs a so called helper, thisoption compiles the helper. If you do not add this module you won't beable to FTP through a firewall or gateway properly.

CONFIG_IP_NF_IPTABLES - Thisoption is required if you want do any kind of filtering,masquerading or NAT. Itadds the whole iptables identification framework to the kernel. Withoutthis you won't be able to do anything at all with iptables.

CONFIG_IP_NF_MATCH_LIMIT - Thismodule isn't exactly required but it's used in the example rc.firewall.txt. This optionprovides the LIMIT match, that adds the possibility to control how manypackets per minute that are to be matched, governed by an appropriaterule. For example, -m limit --limit 3/minute wouldmatch a maximum of 3 packets per minute. This module can also be used toavoid certain Denial of Service attacks.

CONFIG_IP_NF_MATCH_MAC - Thisallows us to match packets based on MACaddresses. Every Ethernet adapter has its own MACaddress. We could for instance block packets based on whatMAC address is used and block a certain computerpretty well since the MAC address very seldomchange. We don't use this option in the rc.firewall.txt example or anywhere else.

CONFIG_IP_NF_MATCH_MARK - This allows us touse a MARK match. For example, if we use the targetMARK we could mark a packet and then depending on if thispacket is marked further on in the table, we can match based on this mark. Thisoption is the actual match MARK, and further down we willdescribe the actual target MARK.

CONFIG_IP_NF_MATCH_MULTIPORT - This moduleallows us to match packets with a whole range of destination ports or sourceports. Normally this wouldn't be possible, but with this match it is.

CONFIG_IP_NF_MATCH_TOS - With this match wecan match packets based on their TOS field.TOS stands for Type Of Service.TOS can also be set by certain rules in themangle table and via the ip/tc commands.

CONFIG_IP_NF_MATCH_TCPMSS -This option adds the possibility for us to matchTCP packets based on theirMSS field.

CONFIG_IP_NF_MATCH_STATE - Thisis one of the biggest news in comparison to ipchains.With this module we can do stateful matching on packets. For example, ifwe have already seen traffic in two directions in aTCP connection, this packet will be counted asESTABLISHED. This module is used extensively in therc.firewall.txtexample.

CONFIG_IP_NF_MATCH_UNCLEAN -This module will add the possibility for us to matchIP, TCP,UDP and ICMP packetsthat don't conform to type or are invalid. We could for example drop thesepackets, but we never know if they are legitimate or not. Note that thismatch is still experimental and might not work perfectly in all cases.

CONFIG_IP_NF_MATCH_OWNER - This option willadd the possibility for us to do matching based on the owner of a socket. Forexample, we can allow only the user root to have Internet access. This modulewas originally just written as an example on what could be done with the newiptables. Note that this match is still experimental andmight not work for everyone.

CONFIG_IP_NF_FILTER - Thismodule will add the basic filter table which willenable you to do IP filtering at all. In thefilter table you'll find theINPUT, FORWARD andOUTPUT chains. This module is required if youplan to do any kind of filtering on packets that you receive and send.

CONFIG_IP_NF_TARGET_REJECT -This target allows us to specify that an ICMPerror message should be sent in reply to incoming packets, instead ofplainly dropping them dead to the floor. Keep in mind thatTCP connections, as opposed toICMP and UDP, are alwaysreset or refused with a TCP RST packet.

CONFIG_IP_NF_TARGET_MIRROR -This allows packets to be bounced back to the sender of the packet. Forexample, if we set up a MIRROR target ondestination port HTTP on ourINPUT chain and someone tries to access thisport, we would bounce his packets back to him and finally he wouldprobably see his own homepage.

CONFIG_IP_NF_NAT - This moduleallows network address translation, orNAT, in its different forms. This option gives usaccess to the nat table in iptables. This option is required if we want todo port forwarding, masquerading, etc. Note that this option is notrequired for firewalling and masquerading of aLAN, but you should have it present unless youare able to provide unique IP addresses for all hosts. Hence, this optionis required for the example rc.firewall.txt script to work properly, and mostdefinitely on your network if you do not have the ability to add unique IPaddresses as specified above.

CONFIG_IP_NF_TARGET_MASQUERADE- This module adds the MASQUERADE target. For instanceif we don't know what IP we have to the Internet this would be thepreferred way of getting the IP instead of usingDNAT or SNAT. In otherwords, if we use DHCP,PPP, SLIP or some otherconnection that assigns us an IP, we need to use this target instead ofSNAT. Masquerading gives a slightly higher loadon the computer than NAT, but will work withoutus knowing the IP address in advance.

CONFIG_IP_NF_TARGET_REDIRECT -This target is useful together with applicationproxies, for example. Instead of letting a packet pass rightthrough, we remap them to go to our local box instead. In other words, wehave the possibility to make a transparent proxythis way.

CONFIG_IP_NF_TARGET_LOG - Thisadds the LOG target and its functionality toiptables. We can use this module to log certain packetsto syslogd and hence see what is happening to the packet. This isinvaluable for security audits, forensics or debugging a script you arewriting.

CONFIG_IP_NF_TARGET_TCPMSS -This option can be used to counter Internet Service Providers and serverswho block ICMP Fragmentation Needed packets. Thiscan result in web-pages not getting through, small mails getting throughwhile larger mails don't, ssh works but scp dies after handshake, etc. Wecan then use the TCPMSS target to overcome thisby clamping our MSS (Maximum Segment Size) to thePMTU (Path Maximum Transmit Unit). This way,we'll be able to handle what the authors of Netfilter them selves call"criminally brain-dead ISPs or servers" in the kernel configuration help.

CONFIG_IP_NF_COMPAT_IPCHAINS -Adds a compatibility mode with the obsolescentipchains. Do not look to this as any real long termsolution for solving migration from Linux 2.2 kernels to 2.4 kernels,since it may well be gone with kernel 2.6.

CONFIG_IP_NF_COMPAT_IPFWADM -Compatibility mode with obsolescent ipfwadm. Definitelydon't look to this as a real long term solution.

As you can see, there is a heap of options. I have brieflyexplained here what kind of extra behaviors you can expect from eachmodule. These are only the options available in a vanilla Linux 2.4.9kernel. If you would like to take a look at more options, I suggest youlook at the patch-o-matic functions inNetfilter user-land which will add heaps of otheroptions in the kernel. POM fixes are additionsthat are supposed to be added in the kernel in the future but has notquite reached the kernel yet. These functions should be added in thefuture, but has not quite made it in yet. This may be for various reasons- such as the patch not being stable yet, to Linus Torvalds being unableto keep up, or not wanting to let the patch in to the mainstream kernelyet since it is still experimental.

You will need the following options compiled into your kernel,or as modules, for the rc.firewall.txt script to work. If you need helpwith the options that the other scripts need, look at the example firewallscripts section.

*   CONFIG_PACKET
*   CONFIG_NETFILTER
*   CONFIG_IP_NF_CONNTRACK
*   CONFIG_IP_NF_FTP
*   CONFIG_IP_NF_IRC
*   CONFIG_IP_NF_IPTABLES
*   CONFIG_IP_NF_FILTER
*   CONFIG_IP_NF_NAT
*   CONFIG_IP_NF_MATCH_STATE
*   CONFIG_IP_NF_TARGET_LOG
*   CONFIG_IP_NF_MATCH_LIMIT
*   CONFIG_IP_NF_TARGET_MASQUERADE

At the very least the above will be required for the rc.firewall.txt script. Inthe other example scripts I will explain what requirements they have intheir respective sections. For now, let's try to stay focused on the mainscript which you should be studying now.

User-land setup

First of all, let's look at how we compile theiptables package. It's important to realize that formost part configuration and compilation of iptables goes hand in hand withthe kernel configuration and compilation. Certain distributions comes withthe iptables package preinstalled, one of these are RedHat. However, in Red Hat it is disabled per default. We will check closeron how to enable it and take a look at other distributions further on inthis chapter.

Compiling the user-land applications

First of all unpack the iptables package.Here, we have used the iptables 1.2.6a package and avanilla 2.4 kernel. Unpack as usual, using bzip2 -cdiptables-1.2.6a.tar.bz2 | tar -xvf - (this can also beaccomplished with the tar -xjvfiptables-1.2.6a.tar.bz2, which should do pretty much the same asthe first command. However, this may not work with older versions oftar). The package should now be unpacked properly intoa directory named iptables-1.2.6a. Formore information read the iptables-1.2.6a/INSTALLfile which contains pretty good information on compiling and getting theprogram to run.

After this, there you have the option of configuring andinstalling extra modules and options etcetera for the kernel.The stepdescribed here will only check and install standard patches that arepending for inclusion to the kernel, there are some even more experimentalpatches further along, which may only be available when you carry outother steps.

Note

Some of these patches are highly experimental and may not be such agood idea to install them. However, there are heaps of extremelyinteresting matches and targets in this installation step so don't beafraid of at least looking at them.

To carry out this step we do something like this from the rootof the iptables package:

make pending-patches KERNEL_DIR=/usr/src/linux/

The variable KERNEL_DIR should point to the actual place thatyour kernel source is located at. Normally this should be/usr/src/linux/ but this may vary, and most probably youwill know yourself where the kernel source is available.

Note

This only asks about certain patches that are just about toenter the kernel anyway. There might be more patches and additions thatthe developers of Netfilter are about to add to the kernel, but is a bitfurther away from actually getting there. One way to install these are bydoing the following:

make most-of-pom KERNEL_DIR=/usr/src/linux/

The above command would ask about installing parts of what inNetfilter world is called patch-o-matic, but still skipthe most extreme patches that might cause havoc in your kernel. Note thatwe say ask, because that's what these commands actually do. They ask youbefore anything is changed in the kernel source. To be able to installall of the patch-o-maticstuff you will need to run the following command:

make patch-o-matic KERNEL_DIR=/usr/src/linux/

Don't forget to read the help for each patch thoroughly before doing anything. Some patches will destroy other patches while others may destroy your kernel if used together with some patches from patch-o-maticetc.

Note

You may totally ignore the above steps if you don't want topatch your kernel, it is in other words not necessary to do the above. However,there are some really interesting things in thepatch-o-matic that you may want to look at so there'snothing bad in just running the commands and see what they contain.

After this you are finished doing thepatch-o-matic parts of installation, you may nowcompile a new kernel making use of the new patches that you have added tothe source. Don't forget to configure the kernel again since the newpatches probably are not added to the configured options. You may waitwith the kernel compilation until after the compilation of the user-landprogram iptables if you feel like it, though.

Continue by compiling the iptables user-land application. Tocompile iptables you issue a simple command that looks likethis:

make KERNEL_DIR=/usr/src/linux/

The user-land application should now compile properly. If not,you are on your own, or you could subscribe to the Netfilter mailing list, where you have thechance of asking for help you with your problems. There are a few thingsthat might go wrong with the installation of iptables,so don't panic if it won't work. Try to think logically about it and findout what's wrong, or get someone to help you.

If everything has worked smoothly, you're ready to install thebinaries by now. To do this, you would issue the following command toinstall them:

make install KERNEL_DIR=/usr/src/linux/

Hopefully everything should work in the program now. To use anyof the changes in the iptables user-land applicationsyou should now recompile and reinstall your kernel and modules, if youhadn't done so before. For more information about installing the user-landapplications from source, check the INSTALL file inthe source which contains excellent information on the subject ofinstallation.

Installation on Red Hat 7.1

Red Hat 7.1 comes preinstalled with a 2.4.x kernel that hasNetfilter and iptablescompiled in. It also contains all the basic user-land programs andconfiguration files that is needed to run it. However, the Red Hat peoplehave disabled the whole thing by using the backward compatibleipchains module. Annoying to say the least, and a lotof people keep asking different mailing lists whyiptables don't work. So, let's take a brief look at howto turn the ipchains module off and how to installiptables instead.

Note

The default Red Hat 7.1 installation today comes with anhopelessly old version of the user-space applications, so you might want tocompile a new version of the applications as well as install a new andcustom compiled kernel before fully exploitingiptables.

First of all you will need to turn off the ipchains modulesso it won't start in the future. To do this, you will need to change somefilenames in the /etc/rc.d/ directory-structure. Thefollowing command should do it:

    chkconfig --level 0123456 ipchains off

By doing this we move all the soft links that points to the/etc/rc.d/init.d/ipchains script to K92ipchains. Thefirst letter which per default would be S, tells the initscripts to startthe script. By changing this to K we tell it to Kill the service instead,or to not run it if it was not previously started. Now the service won'tbe started in the future.

However, to stop the service from actually running right now we need to run another command. This is the service command which can be used to work on currently running services. We would then issue the following command to stop the ipchains service:

service ipchains stop

Finally, to start the iptables service. First of all, we needto know which run-levels we want it to run in. Normally this would be in run-level2, 3 and 5. These run-levels are used for the following things:

* 2. Multiuser without NFS or the same as 3 if there is no networking.
* 3. Full multiuser mode, i.e. the normal run-level to run in.
* 5. X11. This is used if you automatically boot into Xwindows.

To make iptables run in these run-levels we would do thefollowing commands:

chkconfig --level 235 iptables on

The above commands would in other words make the iptablesservice run in run-level 2, 3 and 5. If you'd like theiptables service to run in some other run-level you wouldhave to issue the same command in those. However, none of the other run-levelsshould be used, so you should not really need to activate it for thoserun-levels. Level 1 is for single user mode, i.e, when you need to fix a screwedup box. Level 4 should be unused, and level 6 is for shutting the computer down.

To activate the iptables service, we just run the followingcommand:

service iptables start

There are no rules in the iptables script.To add rules to an Red Hat 7.1 box, there is two common ways. Firstly, youcould edit the /etc/rc.d/init.d/iptables script. Thiswould have the undesired effect of deleting all the rules if you updatedthe iptables package by RPM. The other way would be to load the rule-setand then save it with the iptables-save command andthen have it loaded automatically by the rc.d scripts.

First we will describe the how to set upiptables by cutting and pasting to theiptables init.d script. To add rules that are to be runwhen the computer starts the service, you add them under the start)section, or in the start() function. Note, if you add the rules under thestart) section don't forget to stop the start() function in the start)section from running. Also, don't forget to edit a the stop) sectioneither which tells the script what to do when the computer is going downfor example, or when we are entering a run-level that doesn't requireiptables. Also, don't forget to check out the restartsection and condrestart. Note that all this work will probably be trashedif you have, for example, Red Hat Network automatically update yourpackages. It may also be trashed by updating from theiptables RPM package.

The second way of doing the set up would require the following:First of all, make and write a rule-set in a shell script file, or directly withiptables, that will meet your requirements, and don'tforget to experiment a bit. When you find a set up that works withoutproblems, or as you can see without bugs, use theiptables-save command. You could either use itnormally, i.e. iptables-save > /etc/sysconfig/iptables,which would save the rule-set to the file/etc/sysconfig/iptables. This file is automaticallyused by the iptables rc.d script to restore the rule-setin the future. The other way is to save the script by doingservice iptables save, which would save the scriptautomatically to /etc/sysconfig/iptables. The nexttime you reboot the computer, the iptables rc.d scriptwill use the command iptables-restore to restore therule-set from the save-file /etc/sysconfig/iptables.Do not intermix these two methods, since they may heavily damage eachother and render your firewall configuration useless.

When all of these steps are finished, you can deinstall the currentlyinstalled ipchains and iptables packages.This because we don't want the system to mix up the newiptables user-land application with the old preinstallediptables applications. This step is only necessary if youare going to install iptables from the source package. It's notunusual that the new and the old package to get mixed up, since the rpm basedinstallation installs the package in non-standard places and won't getoverwritten by the installation for the new iptables package.To carry out the deinstallation, do as follows:

rpm -e iptables

And why keep ipchains lying around if youwon't be using it any more? Removing it is done the same way as with theold iptables binaries, etc:

rpm -e ipchains

After all this has been completed, you will have finished withthe update of the iptables package from source, havingfollowed the source installation instructions. None of the old binaries,libraries or include files etc should be lying around any more.

ChapterÂ 3.Â Traversing of tables and chains

Table of Contents

In this chapter we'll discuss how packets traverse the differentchains, and in which order. We will also discuss the order in which thetables are traversed. We'll see how valuable this is later on, when wewrite our own specific rules. We will also look at the points whichcertain other components, that also are kernel dependent, enter into thepicture. Which is to say the different routing decisions and so on. Thisis especially necessary if we want to write iptablesrules that could change routing patterns/rules for packets; i.e. why andhow the packets get routed, good examples of this isDNAT and SNAT. Not to be forgottenare, of course, the TOS bits.

General

When a packet first enters the firewall, it hits the hardwareand then gets passed on to the proper device driver in the kernel. Thenthe packet starts to go through a series of steps in the kernel, before itis either sent to the correct application (locally), or forwarded toanother host - or whatever happens to it.

First, let us have a look at a packet that is destined for ourown local host. It would pass through the following steps before actuallybeing delivered to our application that receives it:

TableÂ 3.1.Â Destination local host (our ownmachine)
StepTableChainComment1Â Â On the wire (e.g., Internet)2Â Â Comes in on the interface (e.g., eth0)3manglePREROUTINGThis chain is normally used for mangling packets, i.e.,changing TOS and so on.4natPREROUTINGThis chain is used for DNAT mainly. Avoid filtering in this chain since it will be bypassed in certain cases.5Â Â Routing decision, i.e., is the packet destined for ourlocal host or to be forwarded and where.6mangleINPUTAt this point, the mangle INPUT chain is hit. We use this chain to mangle packets, after they have been routed, but before they are actually sent to the process on the machine.7filterINPUTThis is where we do filtering for all incoming trafficdestined for our local host. Note that all incoming packets destined for thishost pass through this chain, no matter what interface or in which directionthey came from.8Â Â Local process/application (i.e., server/client program)

Note that this time the packet was passed through theINPUT chain instead of theFORWARD chain. Quite logical. Most probably the onlything that's really logical about the traversing of tables and chains in youreyes in the beginning, but if you continue to think about it, you'll findit will get clearer in time.

Now we look at the outgoing packets from our own local host and what steps theygo through.

Table 3.2. Source local host (our ownmachine)

StepTableChainComment1 Local process/application (i.e., server/clientprogram)2 Routing decision. What source address to use, what outgoinginterface to use, and other necessary information that needs to begathered.3mangleOUTPUTThis is where we mangle packets, it is suggested that you donot filter in this chain since it can have side effects.4natOUTPUTThis chain can be used to NAT outgoing packets from thefirewall itself.5filterOUTPUTThis is where we filter packets going out from the localhost.6manglePOSTROUTINGThe POSTROUTING chain in themangle table is mainly used when we want to do mangling on packets before theyleave our host, but after the actual routing decisions. This chain will be hitby both packets just traversing the firewall, as well as packets created bythe firewall itself.7natPOSTROUTINGThis is where we do SNAT as described earlier. It is suggested that you don't do filtering here since it can have side effects, and certain packets might slip through even though you set a default policy of DROP.8 Goes out on some interface (e.g., eth0)9 On the wire (e.g., Internet)

In this example, we're assuming that the packet is destined for another host onanother network. The packet goes through the different steps in the followingfashion:

Table 3.3. Forwarded packets

StepTableChainComment1 On the wire (i.e., Internet)2 Comes in on the interface (i.e., eth0)3manglePREROUTINGThis chain is normally used for mangling packets, i.e.,changing TOS and so on.4natPREROUTINGThis chain is used for DNAT mainly. SNAT is done further on. Avoid filtering in this chain since it will be bypassed in certain cases.5 Routing decision, i.e., is the packet destined for ourlocal host or to be forwarded and where.6mangleFORWARDThe packet is then sent on to theFORWARD chain of the mangle table. This can be usedfor very specific needs, where we want to mangle the packets after the initialrouting decision, but before the last routing decision made just before thepacket is sent out.7filterFORWARDThe packet gets routed onto theFORWARD chain. Only forwarded packets go throughhere, and here we do all the filtering. Note that all traffic that'sforwarded goes through here (not only in one direction), so you need tothink about it when writing your rule-set.8manglePOSTROUTINGThis chain is used for specific types of packetmangling that we wish to take place after all kinds of routing decisions hasbeen done, but still on this machine.9natPOSTROUTINGThis chain should first and foremost be used forSNAT. Avoid doingfiltering here, since certain packets might pass this chain without everhitting it. This is also where Masquerading is done.10 Goes out on the outgoing interface (i.e., eth1).11 Out on the wire again (i.e., LAN).

As you can see, there are quite a lot of steps to pass through.The packet can be stopped at any of the iptableschains, or anywhere else if it is malformed; however, we are mainlyinterested in the iptables aspect of this lot. Do notethat there are no specific chains or tables for different interfaces oranything like that. FORWARD is always passed byall packets that are forwarded over this firewall/router.

Caution

Do not use the INPUT chain to filter on in the previousscenario! INPUT is meant solelyfor packets to our localhost that do not get routed to any otherdestination.

We have now seen how the different chains are traversed in three separatescenarios. If we were to figure out a good map of all this, it would looksomething like this:

To clarify this image, consider this. If we get a packet into the firstrouting decision that is not destined for the local machine itself, it will berouted through the FORWARD chain. If the packet is,on the other hand, destined for an IP address thatthe local machine is listening to, we would send the packet through theINPUT chain and to the local machine.

Also worth a note, is the fact that packets may be destined for the localmachine, but the destination address may be changed within thePREROUTING chain by doingNAT. Since this takes place before the first routingdecision, the packet will be looked upon after this change. Because of this,the routing may be changed before the routing decision is done. Do note, thatall packets will be going through one or the other pathin this image. If you DNAT a packet back to the samenetwork that it came from, it will still travel through the rest of the chainsuntil it is back out on the network.

Tip

If you feel that you want more information, you could use the rc.test-iptables.txtscript. This test script should give you the necessary rules to test how thetables and chains are traversed.

mangle table

This table should as we've already noted mainly be used for mangling packets.In other words, you may freely use the mangle matches etc that could be usedto change TOS (Type Of Service) fields and so on.

Caution

You are strongly advised not to use this table for anyfiltering; nor will any DNAT, SNATor Masquerading work in this table.

Targets that are only valid in the mangle table:

  * TOS

  * TTL

  * MARK

The TOS target is used to set and/or changethe Type of Service field in the packet. Thiscould be used for setting up policies on the network regarding how apacket should be routed and so on. Note that this has not been perfectedand is not really implemented on the Internet and most of the routersdon't care about the value in this field, and sometimes, they act faultyon what they get. Don't set this in other words for packets going to theInternet unless you want to make routing decisions on it, with iproute2.

The TTL target is used to change theTTL (Time To Live) field of the packet. We couldtell packets to only have a specific TTL and soon. One good reason for this could be that we don't want to give ourselfaway to nosy Internet Service Providers. Some Internet Service Providersdo not like users running multiple computers on one single connection,and there are some Internet Service Providers known to look for a singlehost generating different TTL values, andtake this as one of many signs of multiple computers connected to asingle connection.

The MARK target is used to set specialmark values to the packet. These marks could thenbe recognized by the iproute2 programs to do differentrouting on the packet depending on what mark theyhave, or if they don't have any. We could also do bandwidth limiting andClass Based Queuing based on these marks.

nat table

This table should only be used for NAT(Network Address Translation) on different packets. In other words, it shouldonly be used to translate the packet's source field or destination field. Note that,as we have said before, only the first packet in a stream will hit this chain.After this, the rest of the packets will automatically have the sameaction taken on them as the first packet. The actual targets that dothese kind of things are:

* DNAT

* SNAT

* MASQUERADE

The DNAT target is mainly used in cases where you   have a public IP and want to redirect accesses to the firewall to some   other host (on a DMZ for example). In other words,   we change the destination address of the packet and reroute it to the host.

SNAT is mainly usedfor changing the source address of packets. For the most part you'll hideyour local networks or DMZ, etc. A very good examplewould be that of a firewall of which we know outside IP address, butneed to substitute our local network's IP numbers with that of our firewall.With this target the firewall will automatically SNATand De-SNAT the packets, hence making it possible to makeconnections from the LAN to the Internet. If yournetwork uses 192.168.0.0/netmask for example, the packets would never get backfrom the Internet, because IANA has regulated these networks (amongothers) as private and only for use in isolated LANs.

The MASQUERADE target is used in exactly the same way asSNAT, but the MASQUERADE target takes alittle bit more overhead to compute. The reason for this, is that each timethat the MASQUERADE target gets hit by a packet, itautomatically checks for the IP address to use, instead of doing as theSNAT target does - just using the single configured IPaddress. The MASQUERADE target makes it possible to workproperly with Dynamic DHCP IP addresses that your ISPmight provide for your PPP, PPPoE orSLIP connections to the Internet.

Filter table

The filter table is mainly used forfiltering packets. We can match packets and filter them in whatever way wewant. This is the place that we actually takeaction against packets and look at what they contain andDROP or /ACCEPT them, depending ontheir content. Of course we may also do prior filtering; however, thisparticular table, is the place for which filtering was designed. Almost alltargets are usable in this chain. We will be more prolific about the filter table here; however you now know that this table is the right place to do your main filtering.

ChapterÂ 4.Â The state machine

Table of Contents

This chapter will deal with the state machine and explain it indetail. After reading trough it, you should have a complete understanding ofhow the State machine works. We will also go through a large set of examples onhow states are dealt within the state machine itself. These should clarifyeverything in practice.

Introduction

The state machine is a special part within iptables that should really notbe called the state machine at all, since it is really a connection trackingmachine. However, most people recognize it under the first name. Throughoutthis chapter i will use this names more or less as if they where synonymous.This should not be overly confusing. Connection tracking is done to let theNetfilter framework know the state of a specific connection. Firewalls thatimplement this are generally called stateful firewalls. A stateful firewall isgenerally much more secure than non-stateful firewalls since it allows us towrite much tighter rule-sets.

Within iptables, packets can be related to tracked connectionsin four different so called states. These are known asNEW, ESTABLISHED,RELATED and INVALID. We will discusseach of these in more depth later. With the --statematch we can easily control who or what is allowed to initiate newsessions.

All of the connection tracking is done by special framework within the kernelcalled conntrack. conntrack may be loaded either as a module, or as aninternal part of the kernel itself. Most of the time, we need and want morespecific connection tracking than the default conntrack engine can maintain.Because of this, there are also more specific parts of conntrack that handlesthe TCP, UDP orICMP protocols among others. These modules grabsspecific, unique, information from the packets, so that they may keep track ofeach stream of data. The information that conntrack gathers is then used totell conntrack in which state the stream is currently in. For example,UDP streams are, generally, uniquely identified bytheir destination IP address, source IPaddress, destination port andsource port.

In previous kernels, we had the possibility to turn on and offdefragmentation. However, since iptables and Netfilter were introducedand connection tracking in particular, this option was gotten rid of. Thereason for this is that connection tracking can not work properly withoutdefragmenting packets, and hence defragmenting has been incorporated intoconntrack and is carried out automatically. It can not be turned off,except by turning off connection tracking. Defragmentation is always carriedout if connection tracking is turned on.

All connection tracking is handled in the PREROUTINGchain, except locally generated packets which are handled in theOUTPUT chain. What this means is that iptables willdo all recalculation of states and so on within thePREROUTING chain. If we send the initial packet in astream, the state gets set to NEW within theOUTPUT chain, and when we receive a return packet,the state gets changed in the PREROUTING chain toESTABLISHED, and so on. If the first packet is notoriginated by ourself, the NEW state is set within thePREROUTING chain of course. So, all state changes andcalculations are done within the PREROUTING andOUTPUT chains of the nat table.

The conntrack entries

Let's take a brief look at a conntrack entry and how to read them in/proc/net/ip_conntrack. This gives a list of all thecurrent entries in your conntrack database. If you have theip_conntrack module loaded, a catof /proc/net/ip_conntrack might look like:

```
tcp      6 117 SYN_SENT src=192.168.1.6 dst=192.168.1.9 sport=32775 \
     dport=22 [UNREPLIED] src=192.168.1.9 dst=192.168.1.6 sport=22 \
     dport=32775 use=2
```

This example contains all the information that the conntrack modulemaintains to know which state a specific connection is in. First of all, wehave a protocol, which in this case is tcp. Next, the same valuein normal decimal coding. After this, we see how long this conntrack entry hasto live. This value is set to 117 seconds right now and is decrementedregularly until we see more traffic. This value is then reset to the defaultvalue for the specific state that it is in at that relevant point of time.Next comes the actual state that this entry is in at the present point oftime. In the above mentioned case we are looking at a packet that is in theSYN_SENT state. The internal value of aconnection is slightly different from the ones used externally withiptables. Thevalue SYN_SENT tells us that we are lookingat a connection that has only seen a TCP SYN packetin one direction. Next, we see the source IPaddress, destination IP address,source port and destinationport. At this point we see a specific keyword that tells us thatwe have seen no return traffic for this connection. Lastly, wesee what we expect of return packets. The information details thesource IP address and destination IPaddress (which are both inverted, since the packet isto be directed back to us). The same thing goes for thesource port and destinationport of the connection. These are the values that should be ofany interest to us.

The connection tracking entries may take on a series of different values,all specified in the conntrack headers available inlinux/include/netfilter-ipv4/ip_conntrack*.h files.These values are dependent on which sub-protocol ofIP we use. TCP,UDP or ICMP protocolstake specific default values as specified inlinux/include/netfilter-ipv4/ip_conntrack.h. We willlook closer at this when we look at each of the protocols; however, wewill not use them extensively through this chapter, since they are not usedoutside of the conntrack internals. Also, depending on how this statechanges, the default value of the time until the connection is destroyedwill also change.

Note

Recently there was a new patch made available in iptables patch-o-matic,called tcp-window-tracking. This patch adds, among other things, all ofthe above timeouts to special sysctl variables, which means that they canbe changed on the fly, while the system is still running. Hence, thismakes it unnecessary to recompile the kernel every time you want to changethe timeouts.

 These can be altered via using specific system calls available in
the/proc/sys/net/ipv4/netfilter directory. You shouldin particular look at
the/proc/sys/net/ipv4/netfilter/ip_ct_* variables.

When a connection has seen traffic in both directions, the conntrack entrywill erase the [UNREPLIED] flag, and thenreset it. The entry tells us that the connection has not seen anytraffic in both directions, will be replaced by the[ASSURED] flag, to be found close to the endof the entry. The [ASSURED] flag tells usthat this connection is assured and that it will not be erased if we reach themaximum possible tracked connections. Thus, connections marked as[ASSURED] will not be erased, contrary tothe non assured connections (those not marked as[ASSURED]). How many connections that theconnection tracking table can hold depends upon a variable that can be setthrough the ip-sysctl functions in recent kernels. The default value held bythis entry varies heavily depending on how much memory you have. On 128 MB ofRAM you will get 8192 possible entries, and at 256 MB of RAM, you will get16376 entries. You can read and set your settings through the/proc/sys/net/ipv4/ip_conntrack_max setting.

User-land states

As you have seen, packets may take on several different states within thekernel itself, depending on what protocol we are talking about. However,outside the kernel, we only have the 4 states as described previously. Thesestates can mainly be used in conjunction with the state match which will thenbe able to match packets based on their current connection tracking state. Thevalid states are NEW,ESTABLISHED, RELATED andINVALID states. The following table will briefly explaineach possible state.

TableÂ 4.1.Â User-land states
StateExplanationNEWThe NEW state tells us that the packet isthe first packet that we see. This means that the first packet that the conntrackmodule sees, within a specific connection, will be matched. For example, if we see aSYN packet and it is the first packet in a connectionthat we see, it will match. However, the packet may as well not be aSYN packet and still be consideredNEW. This may lead to certain problems in some instances,but it may also be extremely helpful when we need to pick up lost connectionsfrom other firewalls, or when a connection has already timed out, but inreality is not closed.   ESTABLISHEDThe ESTABLISHED state has seen traffic in bothdirections and will then continuously match those packets.ESTABLISHED connections are fairly easy to understand. Theonly requirement to get into an ESTABLISHED state is thatone host sends a packet, and that it later on gets a reply from the otherhost. The NEW state will upon receipt of the reply packetto or through the firewall change to the ESTABLISHED state.ICMP error messages and redirects etc can also beconsidered as ESTABLISHED, if we have generated a packetthat in turn generated the ICMP message.   RELATEDThe RELATED state is one of the more trickystates. A connection is considered RELATED when it isrelated to another already ESTABLISHED connection. Whatthis means, is that for a connection to be considered asRELATED, we must first have a connection that is consideredESTABLISHED. The ESTABLISHED connectionwill then spawn a connection outside of the main connection. The newly spawnedconnection will then be considered RELATED, if theconntrack module is able to understand that it is RELATED.Some good examples of connections that can be considered asRELATED are the FTP-dataconnections that are considered RELATED to theFTP control port, and theDCC connections issued throughIRC. This could be used to allowICMP replies, FTP transfersand DCC's to work properly through the firewall. Donote that most TCP protocols and someUDP protocols that rely on this mechanism are quitecomplex and send connection information within the payload of theTCP or UDP data segments,and hence require special helper modules to be correctly understood.INVALIDThe INVALID state means that the packet cannot be identified or that it does not have any state. This may be due toseveral reasons, such as the system running out of memory orICMP error messages that do not respond to any knownconnections. Generally, it is a good idea to DROPeverything in this state.

These states can be used together with the --statematch to match packets based on their connection tracking state. This is whatmakes the state machine so incredibly strong and efficient for our firewall.Previously, we often had to open up all ports above 1024 to let all trafficback into our local networks again. With the state machine in place this isnot necessary any longer, since we can now just open up the firewall forreturn traffic and not for all kinds of other traffic.

TCP connections

In this section and the upcoming ones, we will take a closer look at thestates and how they are handled for each of the three basic protocolsTCP, UDP and ICMP. Also, wewill take a closer look at how connections are handled per default, if theycan not be classified as either of these three protocols. We have chosen tostart out with the TCP protocol since it is astateful protocol in itself, and has a lot of interesting details with regardto the state machine in iptables.

A TCP connection is always initiated with the 3-wayhandshake, which establishes and negotiates the actual connection over whichdata will be sent. The whole session is begun with aSYN packet, then a SYN/ACKpacket and finally an ACK packet to acknowledge thewhole session establishment. At this point the connection is established andable to start sending data. The big problem is, how does connection trackinghook up into this? Quite simply really.

As far as the user is concerned, connection tracking works basically thesame for all connection types. Have a look at the picturebelow to see exactly what state the stream enters during the different stagesof the connection. As you can see, the connection tracking code does notreally follow the flow of the TCP connection, fromthe users viewpoint. Once it has seen one packet(theSYN), it considers the connection as NEW. Once itsees the return packet(SYN/ACK), it considers theconnection as ESTABLISHED. If you think about this asecond, you will understand why. With this particular implementation, you canallow NEW and ESTABLISHED packets toleave your local network, only allow ESTABLISHEDconnections back, and that will work perfectly. Conversely, if the connectiontracking machine were to consider the whole connection establishment asNEW, we would never really be able to stop outsideconnections to our local network, since we would have to allowNEW packets back in again. To make things more complicated,there is a number of other internal states that are used forTCP connections inside the kernel, but which are notavailable for us in User-land. Roughly, they follow the state standardsspecified within RFC 793 - Transmission Control Protocol at page 21-23. We will consider these in more detail further along in this section.

As you can see, it is really quite simple, seen from the user's point of view.However, looking at the whole construction from the kernel's point of view,it's a little more difficult. Let's look at an example. Consider exactly howthe connection states change in the/proc/net/ip_conntrack table. The first state is reportedupon receipt of the first SYN packet in a connection.

```
tcp      6 117 SYN_SENT src=192.168.1.5 dst=192.168.1.35 sport=1031 \
     dport=23 [UNREPLIED] src=192.168.1.35 dst=192.168.1.5 sport=23 \
     dport=1031 use=1
```

As you can see from the above entry, we have a precise state in which a SYNpacket has been sent, (the SYN_SENTflag is set), and to which as yet no reply has been sent (witness the[UNREPLIED] flag). The next internal statewill be reached when we see another packet in the other direction.

```
tcp      6 57 SYN_RECV src=192.168.1.5 dst=192.168.1.35 sport=1031 \
    dport=23 src=192.168.1.35 dst=192.168.1.5 sport=23 dport=1031 \
    use=1
```

Now we have received a corresponding SYN/ACK inreturn. As soon as this packet has been received, the state changes onceagain, this time to SYN_RECV.SYN_RECV tells us that the originalSYN was delivered correctly and that theSYN/ACK return packet also got through the firewallproperly. Moreover, this connection tracking entry has now seen traffic inboth directions and is hence considered as having been replied to. This is notexplicit, but rather assumed, as was the[UNREPLIED] flag above. The final step willbe reached once we have seen the final ACK in the3-way handshake.

```
tcp      6 431999 ESTABLISHED src=192.168.1.5 dst=192.168.1.35 \
    sport=1031 dport=23 src=192.168.1.35 dst=192.168.1.5 \
    sport=23 dport=1031 use=1
```

In the last example, we have gotten the final ACK inthe 3-way handshake and the connection has entered theESTABLISHED state, as far as the internal mechanisms ofiptables are aware. After a few more packets, the connection will also become[ASSURED], as shown in the introductionsection of this chapter.

When a TCP connection is closed down, it is done inthe following way and takes the following states.

As you can see, the connection is never really closed until the lastACK is sent. Do note that this picture only describeshow it is closed down under normal circumstances. A connection may also, forexample, be closed by sending a RST(reset), ifthe connection were to be refused. In this case, the connection would beclosed down after a predetermined time.

When the TCP connection has been closed down, theconnection enters the TIME_WAIT state, whichis per default set to 2 minutes. This is used so that all packets that havegotten out of order can still get through our rule-set, even after theconnection has already closed. This is used as a kind of buffer time so thatpackets that have gotten stuck in one or another congested router can stillget to the firewall, or to the other end of the connection.

   If the connection is reset by a RST packet,the state is changed to CLOSE. Thismeans that the connection per default have 10 seconds before the wholeconnection is definitely closed down. RST packets arenot acknowledged in any sense, and will break the connection directly. Thereare also other states than the ones we have told you about so far. Here is thecomplete list of possible states that a TCP streammay take, and their timeout values.

TableÂ 4.2.Â Internal states

StateTimeout valueNONE30 minutesESTABLISHED5 daysSYN_SENT2 minutesSYN_RECV60 secondsFIN_WAIT2 minutesTIME_WAIT2 minutesCLOSE10 secondsCLOSE_WAIT12 hoursLAST_ACK30 secondsLISTEN>2 minutes

These values are most definitely not absolute. They may change with kernelrevisions, and they may also be changed via the proc file-system in the/proc/sys/net/ipv4/netfilter/ip_ct_tcp_* variables. Thedefault values should, however, be fairly well established in practice. Thesevalues are set in jiffies (or 1/100th parts of seconds), so 3000 means 30seconds.

Note

Also note that the User-land side of the state machine does not look atTCP flags set in the TCPpackets. This is generally bad, since you may want to allow packets in theNEW state to get through the firewall, but when you specifythe NEW flag, you will in most cases meanSYN packets.

This is not what happens with the current state implementation; instead, even apacket with no bit set or an ACK flag, will count asNEW and if you match on NEW packets.This can be used for redundant firewalling and so on, but it is generallyextremely bad on your home network, where you only have a single firewall. Toget around this behavior, you could use the command explained in the State NEW packets but no SYN bit set section of the Common problems and questions appendix.Another way is to install the tcp-window-tracking extensionfrom patch-o-matic, which will make the firewall able totrack states depending on the TCP window settings.

UDP connections

UDP connections are in them selves not statefulconnections, but rather stateless. There are several reasons why, mainlybecause they don't contain any connection establishment or connectionclosing; most of all they lack sequencing. Receiving twoUDP datagrams in a specific order does not sayanything about which order in which they where sent. It is, however,still possible to set states on the connections within the kernel. Let's havea look at how a connection can be tracked and how it might look in conntrack.

As you can see, the connection is brought up almost exactly in thesame way as a TCP connection. That is, from theuser-land point of view. Internally, conntrack information looks quite a bitdifferent, but intrinsically the details are the same. First of all, let'shave a look at the entry after the initial UDP packethas been sent.

```
udp      17 20 src=192.168.1.2 dst=192.168.1.5 sport=137 dport=1025 \
    [UNREPLIED] src=192.168.1.5 dst=192.168.1.2 sport=1025 \
    dport=137 use=1
```

As you can see from the first and second values, this is anUDP packet. Thefirst is the protocol name, and the second is protocol number. This is justthe same as for TCP connections. The third valuemarks how many seconds this state entry has to live. After this, we get thevalues of the packet that we have seen and the future expectations of packetsover this connection reaching us from the initiating packet sender. These arethe source, destination, source port and destination port. At thispoint, the [UNREPLIED] flag tells us thatthere's so far been no response to the packet. Finally, we get a brief list ofthe expectations for returning packets. Do note that the latter entries arein reverse order to the first values. The timeout at thispoint is set to 30 seconds, as per default.

```
udp      17 170 src=192.168.1.2 dst=192.168.1.5 sport=137 \
     dport=1025 src=192.168.1.5 dst=192.168.1.2 sport=1025 \
     dport=137 use=1
```

At this point the server has seen a reply to the first packet sent out and theconnection is now considered as ESTABLISHED. This is notshown in the connection tracking, as you can see. The main difference is thatthe [UNREPLIED] flag has now gone. Moreover,the default timeout has changed to 180 seconds - but in this example that'sby now been decremented to 170 seconds - in 10 seconds' time, it will be 160seconds. There's one thing that's missing, though, and can change a bit, andthat is the [ASSURED] flag described above.For the [ASSURED] flag to be set on a trackedconnection, there must have been a small amount of traffic over thatconnection.

```
udp      17 175 src=192.168.1.5 dst=195.22.79.2 sport=1025 \
     dport=53 src=195.22.79.2 dst=192.168.1.5 sport=53 \
     dport=1025 [ASSURED] use=1
```

At this point, the connection has become assured. The connection looksexactly the same as the previous example, except for the[ASSURED] flag. If this connection is notused for 180 seconds, it times out. 180 Seconds is a comparatively low value,but should be sufficient for most use. This value is reset to its full valuefor each packet that matches the same entry and passes through the firewall,just the same as for all of the internal states.

ICMP connections

ICMP packets are far from a stateful stream, sincethey are only used for controlling and should never establish any connections.There are four ICMP types that will generate returnpackets however, and these have 2 different states. TheseICMP messages can take the NEW andESTABLISHED states. The ICMP typeswe are talking about are Echo request andreply, Timestamp request andreply, Information requestand reply and finally Address maskrequest and reply. Out of these, thetimestamp request and informationrequest are obsolete and could most probably just be dropped.However, the Echo messages are used in several setupssuch as pinging hosts. Address mask requests are notused often, but could be useful at times and worth allowing. To getan idea of how this could look, have a look at the following image.

As you can see in the above picture, the host sends an echorequest to the target, which is considered asNEW by the firewall. The target then responds with aecho reply which the firewall considers as stateESTABLISHED. When the first echo request has been seen, thefollowing state entry goes into the ip_conntrack.

```
icmp     1 25 src=192.168.1.6 dst=192.168.1.10 type=8 code=0 \
     id=33029 [UNREPLIED] src=192.168.1.10 dst=192.168.1.6 \
     type=0 code=0 id=33029 use=1
```

This entry looks a little bit different from the standard states forTCP and UDP as you can see.The protocol is there, and the timeout, as well as source and destinationaddresses. The problem comes after that however. We now have 3 new fieldscalled type,code and id.They are not special in any way, the typefield contains the ICMP type and thecode field contains theICMP code. These are all available in ICMP types appendix. The finalid field, contains the ICMPID. Each ICMP packet gets an ID set toit when it is sent, and when the receiver gets theICMP message, it sets the sameID within the new ICMPmessage so that the sender will recognize the reply and will be able toconnect it with the correct ICMP request.

The next field, we once again recognize as the[UNREPLIED] flag, which we have seen before.Just as before, this flag tells us that we are currently looking at aconnection tracking entry that has seen only traffic in one direction.Finally, we see the reply expectation for the replyICMP packet, which is the inversion of the originalsource and destination IP addresses. As for the type and code, these arechanged to the correct values for the return packet, so an echo request ischanged to echo reply and so on. The ICMP ID ispreserved from the request packet.

The reply packet is considered as being ESTABLISHED, as wehave already explained. However, we can know for sure that after theICMP reply, there will be absolutely no more legaltraffic in the same connection. For this reason, the connection tracking entryis destroyed once the reply has traveled all the way through the Netfilterstructure.

In each of the above cases, the request is considered asNEW, while the reply is considered asESTABLISHED. Let's consider this more closely. When thefirewall sees a request packet, it considers it as NEW.When the host sends a reply packet to the request it is consideredESTABLISHED.

Note

Note that this means that the reply packet must match the criterion given bythe connection tracking entry to be considered as established, just as withall other traffic types.

ICMP requests has a default timeout of 30 seconds, which you can change in the/proc/sys/net/ipv4/netfilter/ip_ct_icmp_timeout entry.This should in general be a good timeout value, since it will be able to catchmost packets in transit.

Another hugely important part of ICMP is the factthat it is used to tell the hosts what happened to specificUDP and TCP connections orconnection attempts. For this simple reason, ICMP replies will very often berecognized as RELATED to original connections orconnection attempts. A simple example would be theICMP Host unreachable or ICMP Networkunreachable. These should always be spawned back to our host ifit attempts an unsuccessful connection to some other host, but the network orhost in question could be down, and hence the last router trying to reach thesite in question will reply with an ICMP messagetelling us about it. In this case, the ICMP reply isconsidered as a RELATED packet. The following pictureshould explain how it would look.

In the above example, we send out a SYN packet toa specific address. This is considered as a NEW connectionby the firewall. However, the network the packet is trying to reach isunreachable, so a router returns a network unreachableICMP error to us. The connection tracking code canrecognize this packet as RELATED. thanks to the alreadyadded tracking entry, so the ICMP reply is correctlysent to the client which will then hopefully abort. Meanwhile, the firewallhas destroyed the connection tracking entry since it knows this was an errormessage.

The same behavior as above is experienced with UDPconnections if they run into any problem like the above. AllICMP messages sent in reply toUDP connections are considered asRELATED. Consider the following image.

This time an UDP packet is sent to the host. ThisUDP connection is considered asNEW. However, the network is administratively prohibited bysome firewall or router on the way over. Hence, our firewall receives aICMP Network Prohibited in return. The firewall knowsthat this ICMP error message is related to thealready opened UDP connection and sends it as anRELATED packet to the client. At this point, the firewalldestroys the connection tracking entry, and the client receives theICMP message and should hopefully abort.

Default connections

In certain cases, the conntrack machine does not know how to handle a specificprotocol. This happens if it does not know about that protocol in particular,or doesn't know how it works. In these cases, it goes back to a defaultbehavior. The default behavior is used on, for example,NETBLT, MUX andEGP. This behavior looks pretty much thesame as the UDP connection tracking. The first packetis considered NEW, and reply traffic and so forth isconsidered ESTABLISHED.

When the default behavior is used, all of these packets will attain the samedefault timeout value. This can be set via the/proc/sys/net/ipv4/netfilter/ip_ct_generic_timeoutvariable. The default value here is 600 seconds, or 10 minutes. Depending onwhat traffic you are trying to send over a link that uses the defaultconnection tracking behavior, this might need changing. Especially if you arebouncing traffic through satellites and such, which can take a long time.

Complex protocols and connection tracking

Certain protocols are more complex than others. What this meanswhen it comes to connection tracking, is that such protocols may be harderto track correctly. Good examples of these are theICQ, IRC andFTP protocols. Each and every one of theseprotocols carries information within the actual data payload of thepackets, and hence requires special connection tracking helpers to enableit to function correctly.

Let's take the FTP protocol as thefirst example. The FTP protocol first opens up asingle connection that is called the FTP controlsession. When we issue commands through this session, other ports areopened to carry the rest of the data related to that specific command.These connections can be done in two ways, either actively or passively.When a connection is done actively, the FTPclient sends the server a port and IP address to connect to. After this,the FTP client opens up the port and the serverconnects to that specified port from its own port 20 (known as FTP-Data)and sends the data over it.

The problem here is that the firewall will not know about theseextra connections, since they where negotiated within the actual payloadof the protocol data. Because of this, the firewall will be unable to knowthat it should let the server connect to the client over these specificports.

The solution to this problem is to add a special helper to theconnection tracking module which will scan through the data in the controlconnection for specific syntaxes and information. When it runs into thecorrect information, it will add that specific information asRELATED and the server will be able to track theconnection, thanks to that RELATED entry. Consider thefollowing picture to understand the states when theFTP server has made the connection back to theclient.

Passive FTP works the opposite way. TheFTP client tells the server that it wants somespecific data, upon which the server replies with an IP address to connect toand at what port. The client will, upon receipt of this data, connect to thatspecific port, from its own port 20(the FTP-data port), and get the data inquestion. If you have an FTP server behind yourfirewall, you will in other words require this module in addition to yourstandard iptables modules to let clients on the Internet connect to theFTP server properly. The same goes if you areextremely restrictive to your users, and only want to let them reachHTTP and FTP servers on theInternet and block all other ports. Consider the following image and itsbearing on Passive FTP.

Some conntrack helpers are already available within the kernelitself. More specifically, the FTP andIRC protocols have conntrack helpers as ofwriting this. If you can not find the conntrack helpers that you needwithin the kernel itself, you should have a look at thepatch-o-matic tree within user-land iptables. Thepatch-o-matic tree may contain more conntrackhelpers, such as for the ntalk orH.323 protocols. If they are not available in thepatch-o-matic tree, you have a number of options. Either you can look atthe CVS source of iptables, if it has recently gone into that tree, or youcan contact the Netfilter-develmailing list and ask if it is available. If it is not, and there are noplans for adding it, you are left to your own devices and would mostprobably want to read the RustyRussell's Unreliable Netfilter Hacking HOW-TO which is linked fromthe Other resources and linksappendix.

Conntrack helpers may either be statically compiled into the kernel, or asmodules. If they are compiled as modules, you can load them with the followingcommand

```
modprobe ip_conntrack_*
```

Do note that connection tracking has nothing to do withNAT, and hence you may require more modules if youare NAT'ing connections as well. For example, if you were to want toNAT and track FTPconnections, you would need the NAT module as well.All NAT helpers starts with ip_nat_ and follow thatnaming convention; so for example the FTPNAT helper would be namedip_nat_ftp and the IRCmodule would be named ip_nat_irc. The conntrackhelpers follow the same naming convention, and hence theIRC conntrack helper would be namedip_conntrack_irc, while theFTP conntrack helper would be namedip_conntrack_ftp.

ChapterÂ 5.Â Saving and restoring large rule-sets

Table of Contents
The iptables package comes with two more tools that are very useful, specially if you are dealing with larger rule-sets. These two tools are called iptables-save and iptables-restore and are used to save and restore rule-sets to a specific file-format that looks a quite a bit different from the standard shell code that you will see in the rest of this tutorial.

Speed considerations

One of the largest reasons for using the iptables-save and iptables-restore commands is that they will speed up the loading and saving of larger rule-sets considerably. The main problem with running a shell script that contains iptables rules is that each invocation of iptables within the script will first extract the whole rule-set from the Netfilter kernel space, and after this, it will insert or append rules, or do whatever change to the rule-set that isneeded by this specific command. Finally, it will insert the new rule-set from its own memory into kernel space. Using a shell script, this is done for each and every rule that we want to insert, and for each time we do this, it takes more time to extract and insert the rule-set.

To solve this problem, there is the iptables-save and restore commands. The iptables-save command is used to save the rule-set into a specially formatted text-file, and the iptables-restore command is used to load this text-file into kernel again. The best parts of these commands is that they will load and save the rule-set in one single request. iptables-save will grab the whole rule-set from kernel and save it to a file in one single movement. iptables-restore will upload that specific rule-set to kernel in a single movement for each table. In other words, instead of dropping the rule-set out of kernel some 30.000 times, for really large rule-sets, and then upload it to kernel again that many times, we can now save the whole thing into a file in one movement and then upload the whole thing in as little as three movements depending on how many tables you use.

As you can understand, these tools are definitely something for you if you are working on a huge set of rules that needs to be inserted. However, they do have drawbacks that we will discuss more in the next section.

Drawbacks with restore

As you may have already wondered, can iptables-restore handle any kind of scripting? So far, no, it can not and it will most probably never be able to. This is the main flaw in using iptables-restore since you will not be able todo a huge set of things with these files. For example, what if you have aconnection that has a dynamically assigned IP address and you want to grab this dynamic IP every-time the computer boots up and then use that value within your scripts? With iptables-restore, this is more or less impossible.

One possibility to get around this is to make a small script which grabs the values you would like to use in the script, then sed the iptables-restore file for specific keywords and replace them with the values collected via the small script. At this point, you could save it to a temporary file, and then use iptables-restore to load the new values. This causes a lot of problems however, and you will be unable to use iptables-save properly since it would probably erase your manually added keywords in the restore script. It is in other words a clumsy solution.

Another solution is to load the iptables-restore scripts first, and then load a specific shell script that inserts more dynamic rules in their proper places. Of course, as you can understand, this is just as clumsy as the first solution. iptables-restore is simply not very well suited for configurations where IP addresses are dynamically assigned to your firewall or where you want different behaviors depending on configuration options and so on.

Another drawback with iptables-restore and iptables-save is that it is not fully functional as of writing this. The problem is simply that not a lot of people use it as of today and hence there is not a lot of people finding bugs, and in turn some matches and targets will simply be inserted badly, which may lead to some strange behaviors that you did not expect. Even though these problems exist, I would highly recommend using these tools which should work extremely well for most rule-sets as long as they do not contain some of the new targets or matches that it does not know how to handle properly.

iptables-save

The iptables-save command is, as we have already explained, a tool to save the current rule-set into a file that iptables-restore can use. This command is quite simple really, and takes only two arguments. Take a look at the following example to understand the syntax of the command.

iptables-save [-c] [-t table]

The -c argument tells iptables-save to keep the values specified in the byte and packet counters. This could for example be useful if we would like to reboot our main firewall, but not loose byte and packet counters which we may use for statistical purposes. Issuing a iptables-save command with the -c argument would then make it possible for us to reboot but without breaking our statistical and accounting routines. The default value is, of course, to not keep the counters intact when issuing this command.

The -t argument tells the iptables-save command which tables to save. Withoutthis argument the command will automatically save all tables available into the file. The following is an example on what output you can expect from the iptables-save command if you do not have any rule-set loaded.

```
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:17 2002
*filter
:INPUT ACCEPT [404:19766]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [530:43376]
COMMIT
# Completed on Wed Apr 24 10:19:17 2002
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:17 2002
*mangle
:PREROUTING ACCEPT [451:22060]
:INPUT ACCEPT [451:22060]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [594:47151]
:POSTROUTING ACCEPT [594:47151]
COMMIT
# Completed on Wed Apr 24 10:19:17 2002
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:17 2002
*nat
:PREROUTING ACCEPT [0:0]
:POSTROUTING ACCEPT [3:450]
:OUTPUT ACCEPT [3:450]
COMMIT
# Completed on Wed Apr 24 10:19:17 2002
```

This contains a few comments starting with a # sign. Each table is marked like *>table-name>, for example *mangle. Then within each table we have the chain specifications and rules. A chain specification looks like :>chain-name>
>chain-policy> [>packet-counter>:>byte-counter>]. The chain-name may be
for example PREROUTING, the policy is described previously and can for example be ACCEPT. Finally the packet-counter and byte-counters are the same counters as in the output from iptables -L -v. Finally, each table declaration ends in a COMMIT keyword. The COMMIT keyword tells us that at this point we should commit all rules currently in the pipeline to kernel.

The above example is pretty basic, and hence I believe it is nothing morethan proper to show a brief example which contains a very small Iptables-save ruleset. If we would run iptables-save on this, it would look something like this in the output:

```
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:55 2002
*filter
:INPUT DROP [1:229]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i eth1 -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
-A OUTPUT -m state --state NEW,RELATED,ESTABLISHED -j ACCEPT
COMMIT
# Completed on Wed Apr 24 10:19:55 2002
```

```
# Generated by iptables-save v1.2.6a on Wed Apr 24 10:19:55 2002
*mangle
:PREROUTING ACCEPT [658:32445]
:INPUT ACCEPT [658:32445]
:FORWARD ACCEPT [0:0]
```

As you can see, each command has now been prefixed with the byte and packet counters since we used the -c argument. Except for this, the command-line is quite intact from the script. The only problem now, is how to save the output to a file. Quite simple, and you should already know how to do this if you have used linux at all before. It is only a matter of piping the command output on to the file that you would like to save it as. This could look like the following:

    iptables-save -c > /etc/iptables-save

The above command will in other words save the whole rule-set to a file called /etc/iptables-save with byte and packet counters still intact.

iptables-restore

The iptables-restore command is used to restore the iptables rule-set that was saved with the iptables-save command. It takes all the input from standard input and can not load from files as of writing this, unfortunately. This is the command syntax for iptables-restore:

iptables-restore [-c] [-n]

The -c argument restores the byte and packet counters and must be used if you want to restore counters that was previously saved with iptables-save. This argument may also be written in its long form --counters.

The -n argument tells iptables-restore to not overwrite the previously written rules in the table, or tables, that it is writing to. The default behavior of iptables-restore is to flush and destroy all previously inserted rules. The short -n argument may also be replaced with the longer format --noflush.

To load rule-set with the iptables-restore command, we could do this in several ways, but we will mainly look at the simplest and most common way here.

    cat /etc/iptables-save | iptables-restore -c

This would cat the rule-set located within the /etc/iptables-save file and then pipe it to iptables-restore which takes the rule-set on the standard input and then restores it, including byte and packet counters. It is that simple to begin with. This command could be varied until oblivion and we could show different piping possibilities, however, this is a bit out of the scope of this chapter, and hence we will skip that part and leave it as an exercise for the reader to experiment with.

The rule-set should now be loaded properly to kernel and everything should work. If not, you may possibly have run into a bug in these commands, however likely that sounds.

ChapterÂ 6.Â How a rule is built

Table of Contents

This chapter will discuss at length how to build your own rules. A rule couldbe described as the directions the firewall will adhere to when blocking orpermitting different connections and packets in a specific chain. Each lineyou write that's inserted to a chain should be considered a rule. We will alsodiscuss the basic matches that are available, and how to use them, as well asthe different targets and how we can construct new targets of our own (i.e.,new sub chains).

Basics

As we have already explained, each rule is a line that the kernel looks at tofind out what to do with a packet. If all the criteria - or matches - are met,we perform the target - or jump - instruction. Normally we would write ourrules in a syntax that looks something like this:

iptables [-t table] command [match] [target/jump]

There is nothing that says that the target instruction has to be last functionin the line. However, you would usually adhere to this syntax to get the bestreadability. Anyway, most of the rules you'll see are written in this way.Hence, if you read someone else's script, you'll most likely recognize thesyntax and easily understand the rule.

If you want to use another table than the standard table, you could insertthe table specification at the point at which [table] is specified. However,it is not necessary to state explicitly what table to use, since by defaultiptables uses the filter table onwhich to implement all commands. Neither do you have to specify the table atjust this point in the rule. It could be set pretty much anywhere along theline. However, it is more or less standard to put the table specification atthe beginning.

One thing to think about though: The command should always come first, oralternatively directly after the table specification. We use 'command' to tellthe program what to do, for example to insert a rule or to add a rule to theend of the chain, or to delete a rule. We shall take a further look at thisbelow.

The match is the part of the rule that we send to the kernel that details thespecific character of the packet, what makes it different from all otherpackets. Here we could specify what IP address the packet comes from, fromwhich network interface, the intended IP address, port, protocol or whatever.There is a heap of different matches that we can use that we will look closerat further on in this chapter.

Finally we have the target of the packet. If all the matches are met for apacket, we tell the kernel what to do with it. We could, for example, tell thekernel to send the packet to another chain that we've created ourselves, andwhich is part of this particular table. We could tell the kernel to drop thepacket dead and do no further processing, or we could tell the kernel to senda specified reply to the sender. As with the rest of the content in thissection, we'll look closer at it further on in the chapter.

Tables

The -t option specifies which table to use. Per default, thefilter table is used. We may specify one of thefollowing tables with the -t option. Do note that this is anextremely brief summary of some of the contents of the Traversing of tables and chainschapter.

TableÂ 6.1.Â Tables

TableExplanationnatThe nat table is used mainly forNetwork Address Translation."NAT"ed packets get their IP addresses altered,according to our rules. Packets in a stream only traverse this table once. Weassume that the first packet of a stream is allowed. The rest of thepackets in the same stream are automatically "NAT"edor Masqueraded etc, and will be subject to the same actions as the firstpacket. These will, in other words, not go through this table again, butwill nevertheless be treated like the first packet in the stream. This isthe main reason why you should not do any filtering in this table, whichwe will discuss at greater length further on. ThePREROUTING chain is used to alter packets as soon asthey get in to the firewall. The OUTPUT chain is usedfor altering locally generated packets (i.e., on the firewall) before they getto the routing decision. Finally we have thePOSTROUTING chain which is used to alter packets justas they are about to leave the firewall.mangleThis table is used mainly for mangling packets. Among otherthings, we can change the contents of different packets and that of theirheaders. Examples of this would be to change the TTL,TOS or MARK. Note that theMARK is not really a change to the packet, but a mark valuefor the packet is set in kernel space. Other rules or programs might use thismark further along in the firewall to filter or do advanced routing on; tc isone example. The table consists of five built in chains, thePREROUTING, POSTROUTING, OUTPUT, INPUT and FORWARD chains. PREROUTINGis used for altering packets just as they enter the firewall and before theyhit the routing decision. POSTROUTING is used tomangle packets just after all routing decisions has been made.OUTPUT is used for altering locally generatedpackets before they enter the routing decision. INPUT

is used to alter packets after they have been routed to the local computer itself, but before the user space application actually sees the data. FORWARD is used to mangle packets after they have hit

the first routing decision, but before they actually hit the last routing decision. Note that mangle can not be used for any kind of NetworkAddress Translation or Masquerading, thenat table was made for these kinds of operations.filterThe filter table should be usedexclusively for filtering packets. For example, we couldDROP, LOG, ACCEPT orREJECT packets without problems, as we can in the othertables. There are three chains built in to this table. The first one is namedFORWARD and is used on all non-locally generatedpackets that are not destined for our local host (the firewall, in otherwords). INPUT is used on all packets that aredestined for our local host (the firewall) and OUTPUTis finally used for all locally generated packets.

The above details should have explained the basics about the threedifferent tables that are available. They should be used for totally differentpurposes, and you should know what to use each chain for. If you do notunderstand their usage, you may well dig a pit for yourself in your firewall,into which you will fall as soon as someone finds it and pushes you into it.We have already discussed the requisite tables and chains in more detailwithin the Traversing of tables and chains chapter. If you do not understandthis fully, I advise you to go back and read through it again.

Commands

In this section we will cover all the different commands and what can be donewith them. The command tells iptables what to do with therest of the rule that we send to the parser. Normally we would want either toadd or delete something in some table or another. The following commands areavailable to iptables:

Table 6.2. Commands

Command-A, --appendExampleiptables -A INPUT ...ExplanationThis command appends the rule to the end of thechain. The rule will in other words always be put last in the rule-set andhence be checked last, unless you append more rules later on.Command-D, --deleteExample   iptables -D INPUT --dport 80 -j DROP,   iptables -D INPUT 1 ExplanationThis command deletes a rule in a chain. This could be donein two ways; either by entering the whole rule to match (as in the firstexample), or by specifying the rule number that you want to match. If usethe first method, your entry must match the entry in the chain exactly. If youuse the second method, you must match the number of the rule you want todelete. The rules are numbered from the top of each chain, starting withnumber 1.Command-R, --replaceExampleiptables -R INPUT 1 -s 192.168.0.1 -jDROPExplanationThis command replaces the old entry at the specified line. Itworks in the same way as the --delete command, but instead oftotally deleting the entry, it will replace it with a new entry. The main usefor this might be while you're experimenting with iptables.Command-I, --insertExampleiptables -I INPUT 1 --dport 80 -jACCEPTExplanationInsert a rule somewhere in a chain. The rule is inserted asthe actual number that we specify. In other words, the above example would beinserted as rule 1 in the INPUT chain, and hence fromnow on it would be the very first rule in the chain. Command-L, --listExampleiptables -L INPUTExplanationThis command lists all the entries in the specified chain.In the above case, we would list all the entries in theINPUT chain. It's also legal to not specify any chainat all. In the last case, the command would list all the chains in the specifiedtable (To specify a table, see the Tables section). The exact output is affected byother options sent to the parser, for example the -n and-v options, etc.Command-F, --flushExampleiptables -F INPUTExplanationThis command flushes all rules from the specified chain and isequivalent to deleting each rule one by one, but is quite a bit faster. Thecommand can be used without options, and will then delete all rules in allchains within the specified table.

Command-Z, --zeroExampleiptables -Z INPUTExplanationThis command tells the program to zero all counters in aspecific chain, or in all chains. If you have used the -voption with the -L command, you have probably seen thepacket counter at the beginning of each field. To zero this packet counter,use the -Z option. This option works the same as-L, except that -Z won't list the rules.If -L and -Z is used together (which islegal), the chains will first be listed, and then the packet counters arezeroed.Command-N, --new-chainExampleiptables -N allowedExplanationThis command tells the kernel to create a new chain of thespecified name in the specified table. In the above example we create achain called allowed. Note that there must not already be achain or target of the same name.Command-X, --delete-chainExampleiptables -X allowedExplanationThis command deletes the specified chain from the table. Forthis command to work, there must be no rules that refer to the chain that isto be deleted. In other words, you would have to replace or delete all rulesreferring to the chain before actually deleting the chain. If this command isused without any options, all chains but those built in to the specified tablewill be deleted.Command-P, --policyExampleiptables -P INPUT DROPExplanationThis command tells the kernel to set a specified defaulttarget, or policy, on a chain. All packets that don't match any rule willthen be forced to use the policy of the chain. Legal targets areDROP and ACCEPT(There might be more, mail me if so).Command-E, --rename-chainExampleiptables -E allowed disallowedExplanationThe -E command tellsiptables to change the first name of a chain, to the secondname. In the example above we would, in other words, change the name of thechain from allowed to disallowed. Notethat this will not affect the actual way the table will work. It is, in otherwords, just a cosmetic change to the table.

You should always enter a complete command line, unless you just wantto list the built-in help for iptables or get the version of thecommand. To get the version, use the -v option and to get the helpmessage, use the -h option. As usual, in other words. Nextcomes a few options that can be used with various different commands. Notethat we tell you with which commands the options can be used and what effectthey will have. Also note that we do not include any options here thataffect rules or matches. Instead, we'll take a look at matches andtargets in a later section of this chapter.

TableÂ 6.3.Â Options
Option-v, --verboseCommands used with--list, --append,--insert, --delete,--replaceExplanationThis command gives verbose output and is mainlyused together with the --list command. If used together withthe --list command, it outputs the interface address, ruleoptions and TOS masks. The --list command will also includea bytes and packet counter for each rule, if the --verboseoption is set. These counters uses the K (x1000), M (x1,000,000) and G(x1,000,000,000) multipliers. To overrule this and get exact output, youcan use the -x option, described later. If this option isused with the --append, --insert,--delete or --replace commands, theprogram will output detailed information on how the rule was interpreted andwhether it was inserted correctly, etc.Option-x, --exactCommands used with--listExplanationThis option expands the numerics. The output from--list will in other words not contain the K, M or Gmultipliers. Instead we will get an exact output from the packet and bytecounters of how many packets and bytes that have matched the rule in question.Note that this option is only usable in the --list commandand isn't really relevant for any of the other commands.Option-n, --numericCommands used with--listExplanationThis option tells iptables to output numericalvalues. IP addresses and port numbers will be printed by using theirnumerical values and not host-names, network names or application names.This option is only applicable to the --list command. Thisoption overrides the default of resolving all numerics to hosts and names,where this is possible.Option--line-numbersCommands used with--listExplanationThe --line-numbers command, together withthe --list command, is used to output line numbers. Usingthis option, each rule is output with its number. It could be convenient toknow which rule has which number when inserting rules. This option only workswith the --list command.Option-c, --set-countersCommands used with--insert, --append,--replaceExplanationThis option is used when creating a rule or modifying it insome way. We can then use the option to initialize the packet and bytecounters for the rule. The syntax would be something like--set-counters 20 4000, which would tell the kernel to setthe packet counter to 20 and byte counter to 4000.Option--modprobeCommands used withAllExplanationThe --modprobe option is used to telliptables which module to use when probing for modules oradding them to the kernel. It could be used if yourmodprobe command is not somewhere in the search path etc.In such cases, it might be necessary to specify this option so the programknows what to do in case a needed module is not loaded. This option can beused with all commands.

Matches

In this section we'll talk a bit more about matches. I've chosen tonarrow down the matches into five different subcategories. First of all wehave the generic matches, which can be used in all rules.Then we have the TCP matches which can only be applied toTCP packets. We have UDP matcheswhich can only be applied to UDP packets, andICMP matches which can only be used onICMP packets. Finally we have special matches, suchas the state, owner and limit matches and so on. These final matches have inturn been narrowed down to even more subcategories, even though they mightnot necessarily be different matches at all. I hope this is a reasonablebreakdown and that all people out there can understand it.

Generic matches

This section will deal with Generic matches. Ageneric match is a kind of match that is always available, whatever kind ofprotocol we are working on, or whatever match extensions we have loaded. Nospecial parameters at all are needed to use these matches; in other words. Ihave also included the --protocol match here, even thoughit is more specific to protocol matches. For example, if we want to use aTCP match, we need to use the--protocol match and send TCP asan option to the match. However, --protocol is also a matchin itself, since it can be used to match specific protocols. The followingmatches are always available.

TableÂ 6.4.Â Generic matches
Match-p, --protocolExampleiptables -A INPUT -p tcpExplanationThis match is used to check for certain protocols.Examples of protocols are TCP,UDP and ICMP. The protocol must either be one of the internally specified TCP, UDP or ICMP. It may also take a value specified in the /etc/protocols file, and if it can not find the protocol there it will reply with an error. The protocl may also be a integer value. For example, the ICMP protocol is integer value 1, TCP is 6 and UDP is 17. Finally, it may also take the value ALL. ALL means that it matches only TCP, UDP and ICMP. The command may also take a comma delimited list of protocols, such as udp,tcp which would match allUDP and TCP packets. If thismatch is given the integer value of zero (0), it means ALL protocols, which inturn is the default behavior, if the --protocol match isnot used. This match can also be inversed with the ! sign,so --protocol ! tcp would mean to match UDP and ICMP. Match-s, --src,--sourceExampleiptables -A INPUT -s 192.168.1.1ExplanationThis is the source match, which is used to match packets,based on their source IP address. The main form can be used to match single IPaddresses, such as 192.168.1.1. It could also be usedwith a netmask in a CIDR "bit" form, by specifying the number of ones (1's) onthe left side of the network mask. This means that we could for example add/24 to use a 255.255.255.0 netmask.We could then match whole IP ranges, such as our local networks or networksegments behind the firewall. The line would then look something like192.168.0.0/24. This would match all packets in the192.168.0.x range. Another way is to do it with anregular netmask in the 255.255.255.255 form (i.e.,192.168.0.0/255.255.255.0). We could also invert thematch with an ! just as before. If we were in other wordsto use a match in the form of --source ! 192.168.0.0/24, wewould match all packets with a source address not coming from within the192.168.0.x range. The default is to match all IPaddresses.Match-d, --dst,--destinationExampleiptables -A INPUT -d 192.168.1.1ExplanationThe --destination match is used for packetsbased on their destination address or addresses. It works pretty much the sameas the --source match and has the same syntax, except thatthe match is based on where the packets are going to. To match an IP range, wecan add a netmask either in the exact netmask form, or in the number of ones(1's) counted from the left side of the netmask bits. Examples are:192.168.0.0/255.255.255.0 and192.168.0.0/24. Both of these are equivalent. We couldalso invert the whole match with an ! sign, just as before.--destination ! 192.168.0.1 would in other words match allpackets except those not destined to the 192.168.0.1 IPaddress.Match-i, --in-interfaceExampleiptables -A INPUT -i eth0ExplanationThis match is used for the interface the packet came in on.Note that this option is only legal in the INPUT,FORWARD and PREROUTINGchains and will return an error message when used anywhere else. The defaultbehavior of this match, if no particular interface is specified, is to assumea string value of +. The + value is usedto match a string of letters and numbers. A single + wouldin other words tell the kernel to match all packets without considering whichinterface it came in on. The + string can also be appendedto the type of interface, so eth+ would all Ethernetdevices. We can also invert the meaning of this option with the help of the! sign. The line would then have a syntax looking somethinglike -i ! eth0, which would match all incoming interfaces,except eth0.Match-o, --out-interfaceExampleiptables -A

FORWARD -o eth0ExplanationThe --out-interface match is used forpackets on the interface from which they are leaving. Note that this match isonly available in the OUTPUT,FORWARD and POSTROUTINGchains, the opposite in fact of the --in-interface match.Other than this, it works pretty much the same as the--in-interface match. The + extension isunderstood as

Implicit matches

This section will describe the matches that are loadedimplicitly. Implicit matches are implied, taken forgranted, automatic. For example when we match on --protocoltcp without any further criteria. There are currently three types ofimplicit matches for three different protocols. These are TCPmatches, UDP matches and ICMPmatches. The TCP based matches contain aset of unique criteria that are available only forTCP packets. UDP basedmatches contain another set of criteria that are available only forUDP packets. And the same thing forICMP packets. On the other hand, there can beexplicit matches that are loaded explicitly. Explicitmatches are not implied or automatic, you have to specify themspecifically. For these you use the -m or--match option, which we will discuss in the next section.

TCP matches

 These matches are protocol specific and are only available whenworking with TCP packets and streams. To use thesematches, you need to specify --protocol tcp on the commandline before trying to use them. Note that the --protocoltcp match must be to the left of the protocol specific matches.These matches are loaded implicitly in a sense, just as theUDP and ICMP matches are loadedimplicitly. The other matches will be looked over in the continuation of thissection, after the TCP match section.

TableÂ 6.5.Â TCP matches
Match--sport,--source-portExampleiptables -A INPUT -p tcp --sport 22ExplanationThe --source-port match is used to matchpackets based on their source port. Without it, we imply all source ports.This match can either take a service name or a port number. If you specify aservice name, the service name must be in the /etc/services file, sinceiptables uses this file in which to find. If you specifythe port by its number, the rule will load slightly faster, sinceiptables don't have to check up the service name. However,the match might be a little bit harder to read than if you use the servicename. If you are writing a rule-set consisting of a 200 rules or more, youshould definitely use port numbers, since the difference is reallynoticeable. (On a slow box, this could make as much as 10 seconds'difference, if you have configured a large rule-set containing 1000rules or so). You can also use the --source-port match tomatch any range of ports, --source-port 22:80 for example.This example would match all source ports between 22 and 80. If you omitspecifying the first port, port 0 is assumed (is implicit).--source-port :80 would then match port 0 through 80. Andif the last port specification is omitted, port 65535 is assumed. If you wereto write --source-port 22:, you would have specified amatch for all ports from port 22 through port 65535. If you invert the portrange, iptables automatically reverses your inversion. If you write--source-port 80:22, it is simply interpreted as--source-port 22:80. You can also invert a match by addinga ! sign. For example, --source-port !22 means that you want to match all ports but port 22. The inversioncould also be used together with a port range and would then look like--source-port ! 22:80, which in turn would mean that youwant to match all ports but port 22 through 80. Note that this match does nothandle multiple separated ports and port ranges. For more information aboutthose, look at the multiport match extension.Match--dport,--destination-portExampleiptables -A INPUT -p tcp --dport 22ExplanationThis match is used to match TCPpackets, according to their destination port. It uses exactly the same syntaxas the --source-port match. It understands port and portrange specifications, as well as inversions. It also reverses high and lowports in port range specifications, as above. The match will also assumevalues of 0 and 65535 if the high or low port is left out in a port rangespecification. In other words, exactly the same as the--source-port syntax. Note that this match does not handlemultiple separated ports and port ranges. For more information about those,look at the multiport match extension.Match--tcp-flagsExampleiptables -p tcp --tcp-flags SYN,FIN,ACKSYNExplanationThis match is used to match on theTCP flags in a packet. First of all, the matchtakes a list of flags to compare (a mask) and secondly it takes list offlags that

UDP matches

This section describes matches that will only work togetherwith UDP packets. These matches are implicitlyloaded when you specify the --protocol UDP match andwill be available after this specification. Note thatUDP packets are not connection oriented, andhence there is no such thing as different flags to set in the packet togive data on what the datagram is supposed to do, such as open or closinga connection, or if they are just simply supposed to send data.UDP packets do not require any kind ofacknowledgment either. If they are lost, they are simply lost (Not takingICMP error messaging etc into account). Thismeans that there are quite a lot less matches to work with on aUDP packet than there is onTCP packets. Note that the state machine willwork on all kinds of packets even though UDP orICMP packets are counted as connectionlessprotocols. The state machine works pretty much the same onUDP packets as on TCPpackets.

Table 6.6. UDP matches
Match--sport,--source-portExampleiptables -A INPUT -p udp --sport 53ExplanationThis match works exactly the same as itsTCP counterpart. It is used to perform matches onpackets based on their source UDP ports. It has supportfor port ranges, single ports and port inversions with the same syntax. Tospecify a UDP port range, you could use 22:80 whichwould match UDP ports 22 through 80. If the firstvalue is omitted, port 0 is assumed. If the last port is omitted, port 65535is assumed. If the high port comes before the low port, the ports switch placewith each other automatically. Single UDP portmatches look as in the example above. To invert the port match, add a! sign, --source-port ! 53. This wouldmatch all ports but port 53. The match can understand service names, as longas they are available in the /etc/services file. Note that this match does nothandle multiple separated ports and port ranges. For more information aboutthis, look at the multiport match extension.Match--dport,--destination-portExampleiptables -A INPUT -p udp --dport 53ExplanationThe same goes for this match as for--source-port above. It is exactly the same as for theequivalent TCP match, but here it applies toUDP packets. It matches packets based on theirUDP destination port. The match handles port ranges,single ports and inversions. To match a single port you use, for example,--destination-port 53, to invert this you would use--destination-port ! 53. The first would match allUDP packets going to port 53 while the second wouldmatch packets but those going to the destination port 53. To specify a portrange, you would, for example, use --destination-port 9:19.This example would match all packets destined for UDPport 9 through 19. If the first port is omitted, port 0 is assumed. If thesecond port is omitted, port 65535 is assumed. If the high port is placedbefore the low port, they automatically switch place, so the low port winds upbefore the high port. Note that this match does not handle multiple ports andport ranges. For more information about this, look at the multiport matchextension.

ICMP matches

These are the ICMP matches. These packets areeven more ephemeral, that is to say short lived, thanUDP packets, in the sense that they areconnectionless. The ICMP protocol is mainly used forerror reporting and for connection controlling and suchlike.ICMP is not a protocol subordinated to the IPprotocol,but more of a protocol that augments the IP protocol and helps in handlingerrors. The headers of ICMP packets are very similarto those of the IP headers, but differ in a number of ways. The main featureof this protocol is the type header, that tells us what the packet is for. Oneexample is, if we try to access an unaccessible IP address, we would normallyget an ICMP host unreachable in return. For acomplete listing of ICMP types, see the ICMP types appendix. There is onlyone ICMP specific match available forICMP packets, and hopefully this should suffice. Thismatch is implicitly loaded when we use the --protocol ICMPmatch and we get access to it automatically. Note that all the generic matchescan also be used, so that among other things we can match on the source

TableÂ 6.7.Â ICMP matches

Match--icmp-typeExampleiptables -A INPUT -p icmp --icmp-type8ExplanationThis match is used to specify the ICMPtype to match. ICMP types can bespecifiedeither by their numeric values or by their names. Numerical values arespecifiedin RFC 792. To find a complete listing of the ICMPnamevalues, do an iptables --protocol icmp --help, or check theICMP types appendix. Thismatch can also be inverted with the ! sign in this,--icmp-type ! 8, fashion. Note that some ICMPtypes are obsolete, and others again may be "dangerous" for anunprotected host since they may, among other things, redirect packets to thewrong places.

Explicit matches

Explicit matches are those that have to be specifically loaded withthe -m or --match option. Statematches, for example, demand the directive -m state priorto entering the actual match that you want to use. Some of these matches maybe protocol specific . Some may be unconnected with any specific protocol -for example connection states. These might be NEW (thefirst packet of an as yet unestablished connection),ESTABLISHED (a connection that is already registered in thekernel), RELATED (a new connection that was created by anolder, established one) etc. A few may just have been evolved for testing orexperimental purposes, or just to illustrate what iptables is capable of. Thisin turn means that not all of these matches may at first sight be of any use.Nevertheless, it may well be that you personally will find a use for specificexplicit matches. And there are new ones coming along all the time, with eachnew iptables release. Whether you find a use for them ornot depends on your imagination and your needs. The difference betweenimplicitly loaded matches and explicitly loaded ones, is that the implicitlyloaded matches will automatically be loaded when, for example, you match onthe properties of TCP packets, while explicitlyloaded matches will never be loaded automatically - it is up to you todiscover and activate explicit matches.

Limit match

  The limit match extension must be loadedexplicitly with the -m limit option. This match can,for example, be used to advantage to give limited logging of specificrules etc. For example, you could use this to match all packets that doesnot exceed a given value, and after this value has been exceeded,limit logging of the event in question. Think of a timelimit : You could limit how many times a certain rule may be matched in acertain time frame, for example to lessen the effects ofDoS syn flood attacks. This is its main usage, butthere are more usages, of course. The limit match mayalso be inverted by adding a ! flag in front of thelimit match. It would then be expressed as -m limit !--limit 5/s.This means that all packets will be matched after they have broken thelimit.

  To further explain the limit match, it is basically a tokenbucket filter. Consider having a leaky bucket where the bucket leaks Xpackets per time-unit. X is defined depending on how many matching packetswe get, so if we get 3 packets, the bucket leaks 3 packets per thattime-unit. The --limit option tells us how many packetsto refill the bucket with per time-unit, while the--limit-burst option tells us how big the bucket is inthe first place. So, setting --limit 3/minute --limit-burst5, and then receiving 5 matches will empty the bucket. After 20seconds, the bucket is refilled with another token, and so on until the--limit-burst is reached again or until they get used.

Consider the example below for further explanation of how this may look.

* We set a rule with -m limit --limit 5/second --limit-burst 10/second.The limit-burst token bucket is set to 10 initially. Each packet that matchesthe rule uses a token.

* We get packet that matches, 1-2-3-4-5-6-7-8-9-10, all within a 1/1000 of a second.

* The token bucket is now empty. Once the token bucket is empty, the packetsthat qualify for the rule otherwise no longer match the rule and proceed tothe next rule if any, or hit the chain policy.

* For each 1/5 s without a matching packet, the token count goes up by 1, upto amaximum of 10. 1 second after receiving the 10 packets, we will once again have5 tokens left.

* And of course, the bucket will be emptied by 1 token for each packet it receives.

TableÂ 6.8.Â Limit match options

Match--limitExampleiptables -A INPUT -m limit --limit3/hourExplanationThis sets the maximum average match rate for thelimit match. You specify it with a number and an optionaltime unit. The following time units are currently recognized:/second /minute /hour/day. The default value here is 3 per hour, or3/hour. This tells the limit match howmany times to allow the match to occur per time unit (e.g. perminute).Match--limit-burstExampleiptables -A INPUT -m limit --limit-burst5ExplanationThis is the setting for the burstlimit of the limit match. It tellsiptables the maximum number of packets to match within thegiven time unit. This number gets decremented by one for every time unit(specified by the --limit option) in which the event doesnot occur, back down to the lowest possible value, 1. If the event isrepeated, the counter is again incremented, until the count reaches the burstlimit. And so on. The default --limit-burst value is 5. Fora simple way of checking out how this works, you can use the example Limit-match.txt one-rule-script. Using this script, you can see for yourself how the limit rule works, by simply sending ping packets at different intervals and in different burst numbers. All echo replies will be blocked until the threshold for the burst limit has again been reached.

MAC match

The MAC (Ethernet MediaAccess Control) match can be used to match packets based on theirMAC source address. As of writing this documentation,this match is a little bit limited, however, in the future this may be moreevolved and may be more useful. This match can be used to match packets on thesource MAC address only as previously said.

Note

Do note that to use this module we explicitly load it with the -mmac option. The reason that I am saying this is that a lot of peoplewonder if it should not be -m mac-source, which it shouldnot.

TableÂ 6.9.Â MAC match options

Match--mac-sourceExampleiptables -A INPUT -m mac --mac-source00:00:00:00:00:01ExplanationThis match is used to match packets based on theirMAC source address. The MACaddress specified must be in the formXX:XX:XX:XX:XX:XX, else it will not be legal. The matchmaybe reversed with an ! sign and would look like--mac-source ! 00:00:00:00:00:01. This would in other wordsreverse the meaning of the match, so that all packets except packets from thisMAC address would be matched. Note that sinceMAC addresses are only used on Ethernet typenetworks, this match will only be possible to use for Ethernet interfaces. TheMAC match is only valid in thePREROUTING, FORWARD andINPUT chains and nowhere else.

Mark match

The mark match extension is used to match packets based onthe marks they have set. A mark is a special field, onlymaintained within the kernel, that is associated with the packets as theytravel through the computer. Marks may be used by different kernelroutines for such tasks as traffic shaping and filtering. As of today, thereis only one way of setting a mark in Linux, namely theMARK target in iptables. This waspreviously done with the FWMARK target inipchains, and this is why people still refer toFWMARK in advanced routing areas. The mark field iscurrently set to an unsigned integer, or 4294967296 possible values on a32 bit system. In other words, you are probably not going to run into thislimit for quite some time.

Table 6.10. Mark match options
Match--markExampleiptables -t mangle -A INPUT -m mark --mark1ExplanationThis match is used to match packets that have previouslybeen marked. Marks can be set with the MARK targetwhich we will discuss in the next section. All packets traveling throughNetfilter get a special markfield associated with them. Note that this markfield is not in any way propagated, within or outside thepacket. It stays inside the computer that made it. If the markfield matches the mark, it is a match. The markfield is an unsigned integer, hence there can be a maximum of4294967296 different marks. You may also use a mask with the mark. Themark specification would then look like, for example, --mark1/1. If a mask is specified, it is logically AND ed with the markspecified before the actual comparison.

Multiport match

The multiport match extension can be used to specifymultipledestination ports and port ranges. Without the possibility this matchgives, you would have to use multiple rules of the same type, just tomatch different ports.

Note

You can not use both standard port matching and multiport matching at thesame time, for example you can't write: --sport 1024:63353 -mmultiport --dport 21,23,80. This will simply not work. What in facthappens, if you do, is that iptables honors the first element in the rule, andignores the multiport instruction.

Table 6.11. Multiport match options
Match--source-portExampleiptables -A INPUT -p tcp -m multiport
--source-port22,53,80,110ExplanationThis match matches multiple source ports. A maximum of 15separate ports may be specified. The ports must be comma delimited, as in theabove example. The match may only be used in conjunction with the -ptcp or -p udp matches. It is mainly an enhancedversion of the normal --source-port
match.Match--destination-portExampleiptables -A INPUT -p tcp -m
multiport--destination-port22,53,80,110ExplanationThis match is used to match multiple destination ports. Itworks exactly the same way as the above mentioned source port match, exceptthat it matches destination ports. It too has a limit of 15 ports and may onlybe used in conjunction with -p tcp and -pudp.Match--portExampleiptables -A INPUT -p tcp -m multiport
--port22,53,80,110ExplanationThis match extension can be used to match packets basedboth on their destination port and their source port. It works the sameway as the --source-port and--destination-port matches above. It can take a maximum of15 ports and can only be used in conjunction with -p tcpand -p udp. Note that the --port matchwill only match packets coming in from and going to the same port, forexample, port 80 to port 80, port 110 to port 110 and so on.

Owner match

The owner match extension is used to match packets basedon the identity of the process that created them. The ownercan be specified as the process ID either of the user who issued the commandin question, that of the group, the process, the session, or that of thecommand itself. This extension was originally written as an example of whatiptables could be used for. The ownermatch only works within the OUTPUT chain, for obviousreasons: It is pretty much impossible to find out any information about theidentity of the instance that sent a packet from the other end, or where thereis an intermediate hop to the real destination. Even within theOUTPUT chain it is not very reliable, since certainpackets may not have an owner. Notorious packets of that sort are (amongother things) the different ICMP responses.ICMP responses will never match.

Table 6.12. Owner match options
Match--uid-ownerExampleiptables -A OUTPUT -m owner --uid-owner500ExplanationThis packet match will match if the packet was created bythe given User ID (UID). This could be usedtomatch outgoing packets based on who created them. One possible use would be toblock any other user than root from opening new connections outside yourfirewall. Another possible use could be to block everyone but thehttp user from sending packets from theHTTP port. Match--gid-ownerExampleiptables -A OUTPUT -m owner --gid-owner0ExplanationThis match is used to match all packets based on theirGroup ID (GID). This means that we match allpackets based on what group the user creating the packets are in. This couldbeused to block all but the users in thenetwork group from getting out onto theInternet or, as described above, only to allow members of thehttp group to create packets going out fromthe HTTP port. Match--pid-ownerExampleiptables -A OUTPUT -m owner --pid-owner78ExplanationThis match is used to match packets based on theProcess ID (PID) that wasresponsible for them. This match is a bit harder to use, but one example wouldbe only to allow PID 94 to send packets from theHTTP port (if the HTTPprocess is not threaded, of course). Alternatively we could write a smallscript that grabs the PID from aps output for a specific daemon and then adds a rule forit. For an example, you could have a rule as shown in the Pid-owner.txt example. Match--sid-ownerExampleiptables -A OUTPUT -m owner --sid-owner100ExplanationThis match is used to match packets based on theSession ID used by the program in question. The valueof the SID, or Session ID ofa process, is that of the process itself and all processes resulting from theoriginating process. These latter could be threads, or a child of the originalprocess. So, for example, all of our HTTPD processesshould have the same SID as their parent process (theoriginating HTTPD process), if ourHTTPD is threaded (mostHTTPDs are, Apache and Roxen for instance). To showthis in example, we have created a small script called Sid-owner.txt. This script could possibly be run every hour or so together with some extra code to check if the HTTPD is actually running and start it again if necessary, then flush and re-enter our OUTPUT chain if needed.

State match

The state match extension is used in conjunction with theconnection tracking code in the kernel. The state match accesses the connectiontracking state of the packets from the conntracking machine. This allows us toknow in what state the connection is, and works for pretty much all protocols,including stateless protocols such as ICMP and UDP. In all cases,there will be a default timeout for the connection and it will then be droppedfrom the connection tracking database. This match needs to be loadedexplicitly by adding a -m state statement to the rule. Youwill then have access to one new match called state. The concept of statematching is covered more fully in the The state machine chapter, since it is such a large topic.

TableÂ 6.13.Â State matches
Match--stateExampleiptables -A INPUT -m state --stateRELATED,ESTABLISHED
ExplanationThis match option tells the state match whatstates the packets must be in to be matched. There are currently 4 states thatcan be used. INVALID, ESTABLISHED,NEW and RELATED.INVALID means that the packet is associated with no knownstream or connection and that it may contain faulty data or headers.ESTABLISHED means that the packet is part of an alreadyestablished connection that has seen packets in both directions and is fullyvalid. NEW means that the packet has or will start a newconnection, or that it is associated with a connection that has not seenpacketsin both directions. Finally, RELATED means that the packetisstarting a new connection and is associated with an already establishedconnection. This could for example mean an FTP datatransfer, or an ICMP error associatedwithan TCP or UDP connection. Note that theNEW state does not look for SYNbits in TCP packets trying to start a new connectionand should, hence, not be used unmodified in cases where we have only onefirewall and no load balancing between different firewalls. However, there maybe times where this could be useful. For more information on how this could beused, read the The state machine chapter.

TOS match

The TOS match can be used to match packets based on theirTOS field. TOS stands forType Of Service, consists of 8 bits, and is locatedinthe IP header. This match is loaded explicitly by adding -mtos to the rule. TOS is normally used toinform intermediate hosts of the precedence of the stream and its content (itdoesn't really, but it informs of any specific requirements for the stream,such as it having to be sent as fast as possible, or it needing to be able tosend as much payload as possible). How different routers and administratorsdeal with these values depends. Most do not care at all, while others trytheir best to do something good with the packets in question and the data theyprovide.

TableÂ 6.14.Â TOS matches
Match--tosExampleiptables -A INPUT -p tcp -m tos --tos0x16ExplanationThis match is used as described above. It can match packetsbased on their TOS field and their value. This couldbe used, among other things together with the iproute2and advanced routing functions in Linux, to mark packetsfor later usage. The match takes an hex or numeric value as an option, orpossibly one of the names resulting from 'iptables -m tos-h'. At the time of writing it contained the following named values:Minimize-Delay 16 (0x10),Maximize-Throughput 8 (0x08),Maximize-Reliability 4 (0x04),Minimize-Cost 2 (0x02), andNormal-Service 0 (0x00).Minimize-Delay means to minimize the delay inputting the packets through - example of standard services that would requirethis include telnet, SSH andFTP-control.Maximize-Throughput means to find a path thatallows as big a throughput as possible - a standard protocol would beFTP-data.Maximize-Reliability means to maximize thereliability of the connection and to use lines that are as reliable aspossible - a couple of typical examples are BOOTP andTFTP. Minimize-Costmeans minimizing the cost of packets getting through each link to the clientor server; for example finding the route that costs the least to travel along.Examples of normal protocols that would use this would beRTSP (Real Time Stream Control Protocol) and otherstreaming video/radio protocols. Finally,Normal-Service would mean any normal protocolthat has no special needs.

TTL match

The TTL match is used to match packets based on theirTTL (Time To Live) field residing in the IP headers.The TTL field contains 8 bits of data and isdecremented once every time it is processed by an intermediate host betweenthe client and recipient host. If the TTL reaches 0,an ICMP type 11 code 0 (TTL equals 0 during transit)or code 1 (TTL equals 0 during reassembly) is transmitted to the party sendingthe packet and informing it of the problem. This match is only used to matchpackets based on their TTL, and not to changeanything. The latter, incidentally, applies to all kinds of matches. To loadthis match, you need to add an -m ttl to the rule.

TableÂ 6.15.Â TTL matches

Match--ttlExampleiptables -A OUTPUT -m ttl --ttl 60ExplanationThis match option is used to specify theTTL value to match. It takes a numeric value andmatches this value within the packet. There is no inversion and there are noother specifics to match. It could, for example, be used for debugging yourlocal network - e.g. LAN hosts that seem to have problems connecting to hostson the Internet - or to find possible ingress by Trojans etc. The usage isrelatively limited, however; its usefulness really depends on yourimagination. One example would be to find hosts with bad defaultTTL values (could be due to a badly implementedTCP/IP stack, or simply to misconfiguration).

Unclean match

The unclean match takes no options and requiresno more than explicitly loading it when you want to use it. Note that thisoption is regarded as experimental and may not work at all times, nor will ittake care of all unclean packages or problems. The unclean match tries tomatch packets that seem malformed or unusual, such as packets with bad headersor checksums and so on. This could be used to DROPconnections and to check for bad streams, for example; however you should beaware that it could possibly break legal connections.

Targets/Jumps

The target/jumps tells the rule what to do with a packet that is a perfectmatch with the match section of the rule. There are a couple of basic targets,the ACCEPT and DROP targets, which wewill deal with first. However, before we do that, let us have a brief look athow a jump is done.

The jump specification is done in exactly the same way as in the targetdefinition, except that it requires a chain within the same table to jump to.To jump to a specific chain, it is of course a prerequisite that that chainexists. As we have already explained, a user-defined chain is created with the-N command. For example, let's say we create a chain in thefilter table called tcp_packets, like this:

```
iptables -N tcp_packets
```

We could then add a jump target to it like this:

```
iptables -A INPUT -p tcp -j tcp_packets
```

We would then jump from the INPUT chain to thetcp_packets chain and start traversing that chain. When/Ifwe reach the end of that chain, we get dropped back to theINPUT chain and the packet starts traversing from the ruleone step below where it jumped to the other chain (tcp_packets in this case).If a packet is ACCEPTed within one of the sub chains, itwill be ACCEPT'ed in the superset chain also and it willnot traverse any of the superset chains any further. However, do note that thepacket will traverse all other chains in the other tables in a normal fashion.For more information on table and chain traversing, see the Traversing of tables and chainschapter.

 Targets on the other hand specify an action to take on thepacket in question. We could for example, DROP orACCEPT the packet depending on what we want to do.There are also a number of other actions we may want to take, which wewill describe further on in this section. Jumping to targets may incurdifferent results, as it were. Some targets will cause the packet to stoptraversing that specific chain and superior chains as described above.Good examples of such rules are DROP andACCEPT. Rules that are stopped, will not pass throughany of the rules further on in the chain or in superior chains. Othertargets, may take an action on the packet, after which the packet willcontinue passing through the rest of the rules. A good example of thiswould be the LOG, ULOG andTOS targets. These targets can log thepackets, mangle them and then pass them on to the otherrules in the same set of chains. We might, for example, want this so thatwe in addition can mangle both the TTL and theTOS values of a specific packet/stream. Sometargets will accept extra options (What TOS valueto use etc), while others don't necessarily need any options - but wecan include them if we want to (log prefixes, masquerade-to ports and soon). We will try to cover all of these points as we go through the targetdescriptions. Let us have a look at what kinds of targets there are.

ACCEPT target

 This target needs no further options. As soon as the matchspecification for a packet has been fully satisfied, and we specify ACCEPT asthe target, the rule is accepted and will not continue traversing the currentchain or any other ones in the same table. Note however, that a packet thatwas accepted in one chain might still travel through chains within othertables, and could still be dropped there. There is nothing special about thistarget whatsoever, and it does not require, nor have the possibility of,adding options to the target. To use this target, we simply specify-j ACCEPT.

DNAT target

    The DNAT target is used to do DestinationNetwork Address Translation, which means that it is used torewritethe Destination IP address of a packet. If a packet ismatched, and this is the target of the rule, the packet, and all subsequentpackets in the same stream will be translated, and then routed on to thecorrectdevice, host or network. This target can be extremely useful, for example,whenyou have an host running your web server inside a LAN,but no real IP to give it that will work on the Internet. You could then tellthe firewall to forward all packets going to its ownHTTP port, on to the real web server within theLAN. We may also specify a whole range of destination IPaddresses, and the DNAT mechanism will choose thedestination IP address at random for each stream. Hence, we will be able todeal with a kind of load balancing by doing this.

Note that the DNAT target is only available within thePREROUTING and OUTPUT chainsin the nat table, and any of the chains called uponfrom anyof those listed chains. Note that chains containing DNATtargets may not be used from any other chains, such as thePOSTROUTING chain.

TableÂ 6.16.Â DNAT target
Option--to-destinationExampleiptables -t nat -A PREROUTING -p tcp -d 15.45.23.67--dport 80 -j DNAT --to-destination 192.168.1.1-192.168.1.10   ExplanationThe --to-destination option tells theDNAT mechanism which DestinationIP to set in the IP header, and where to send packets that arematched. The above example would send on all packets destined for IP address15.45.23.67 to a range of LAN IP's, namely 192.168.1.1through 10. Note, as described previously, that a single stream will alwaysuse the same host, and that each stream will randomly be given an IP addressthat it will always be Destined for, within that stream. We could also havespecified only one IP address, in which case we would always be connected tothe same host. Also note that we may add a port or port range to which thetraffic would be redirected to. This is done by adding, for example, an :80statement to the IP addresses to which we want toDNAT the packets. A rule could then look like--to-destination 192.168.1.1:80 for example, or like--to-destination 192.168.1.1:80-100 if we wanted to specifya port range. As you can see, the syntax is pretty much the same for theDNAT target, as for the SNAT target eventhough they do two totally different things. Do note that port specificationsare only valid for rules that specify the TCP or UDP protocols with the--protocol option.

 Since DNAT requires quite a lot of work towork properly, I have decided to add a larger explanation on how to workwith it. Let's take a brief example on how things would be done normally.We want to publish our website via our Internet connection. We only haveone IP address, and the HTTP server is located onour internal network. Our firewall has the external IP address$INET_IP, and our HTTP serverhas the internal IP address $HTTP_IP and finally thefirewall has the internal IP address $LAN_IP. The firstthing to do is to add the following simple rule to thePREROUTING chain in the nat table:

```
iptables -t nat -A PREROUTING --dst $INET_IP -p tcp --dport 80 -j DNAT \
--to-destination $HTTP_IP
```

 Now, all packets from the Internet going to port 80 on ourfirewall are redirected (or DNAT'ed) to our internalHTTP server. If you test this from the Internet,everything should work just perfect. So, what happens if you tryconnecting from a host on the same local network as theHTTP server? It will simply not work. This is aproblem with routing really. We start out by dissect what happens in anormal case. The external box has IP address $EXT_BOX,to maintain readability.

* Packet leaves the connecting host going to$INET_IP and source $EXT_BOX.

* Packet reaches the firewall.

* Firewall DNAT's the packet and runs the packetthrough all different chains etcetera.

* Packet leaves the firewall and travels to the $HTTP_IP.

* Packet reaches the HTTP server, and theHTTP box replies back through the firewall, ifthat is the box that the routing database has entered as the gateway for$EXT_BOX. Normally, this would be the default gatewayof the HTTP server.

* Firewall Un-DNAT's the packet again, so the packetlooks as if it was replied to from the firewall itself.

* Reply packet travels as usual back to the client$EXT_BOX.

Now, we will consider what happens if the packet was insteadgenerated by a client on the same network as theHTTP server itself. The client has the IP address$LAN_BOX, while the rest of the machines maintain thesame settings.

* Packet leaves $LAN_BOX to $INET_IP.

* The packet reaches the firewall.

* The packet gets DNAT'ed, and all other requiredactions are taken, however, the packet is not SNAT'ed,so the same source IP address is used on the packet.

* The packet leaves the firewall and reaches theHTTP server.

* The HTTP server tries to respond to thepacket, and sees in the routing databases that the packet came from alocal box on the same network, and hence tries to send the packet directlyto the original source IP address (which nowbecomes the destination IP address).

* The packet reaches the client, and the client gets confused since thereturn packet does not come from the host that it sent the originalrequest to. Hence, the client drops the reply packet, and waits for the"real" reply.

The simple solution to this problem is toSNAT all packets entering the firewall and leaving fora host or IP that we know we do DNAT to. For example,consider the above rule. We SNAT the packetsentering our firewall that are destined for $HTTP_IPport 80 so that they look as if they came from $LAN_IP.This will force the HTTP server to send thepackets back to our firewall, which Un-DNAT's thepackets and sends them on to the client. The rule would look somethinglike this:

```
iptables -t nat -A POSTROUTING -p tcp --dst $HTTP_IP --dport 80 -j SNAT \
--to-source $LAN_IP
```

Remember that the POSTROUTING chain isprocessed last of the chains, and hence the packet will already beDNAT'ed once it reaches that specific chain. This isthe reason that we match the packets based on the internal address.

Warning

This last rule will seriously harm your logging, so it is really advisable not to use this method, but the whole example is still a valid one for all of those who can't afford to set up a specific DMZ or alike. What will happen is this, packet comes from the Internet, gets SNAT'ed and DNAT'ed, and finally hits the HTTP server (for example). The HTTP server now only sees the request as if it was coming from the firewall, and hence logs all requests from the internet as if they came from the firewall.

This can also have even more severe implications. Take a SMTP server on the LAN, that allows requests from the internal network, and you have your firewall set up to forward SMTP traffic to it. You have now effectively created an open relay SMTP server, with horrenduously bad logging!

You will in other words be better off solving these problems by either setting up a separate DNS server for your LAN, or to actually set up a separate DMZ, the latter being preferred if you have the money.

You think this should be enough by now, and it really is,unless considering one final aspect to this whole scenario. What if thefirewall itself tries to access the HTTP server,where will it go? As it looks now, it will unfortunately try to get to itsown HTTP server, and not the server residing on$HTTP_IP. To get around this, we need to add aDNAT rule in the OUTPUT chainas well. Following the above example, this should look something like thefollowing:

```
iptables -t nat -A OUTPUT --dst $INET_IP -p tcp --dport 80 -j DNAT \
--to-destination $HTTP_IP
```

Adding this final rule should get everything up and running.All separate networks that do not sit on the same net as theHTTP server will run smoothly, all hosts on thesame network as the HTTP server will be able toconnect and finally, the firewall will be able to do proper connections aswell. Now everything works and no problems should arise.

Note

Everyone should realize that these rules only effects how the packet isDNAT'ed andSNAT'ed properly.In addition to these rules, you may also need extra rules in the filtertable (FORWARD chain) to allow the packets totraverse through those chains as well. Don't forget that all packets havealready gone through the PREROUTING chain, andshould hence have their destination addresses rewritten already byDNAT.

DROP target

The DROP target does just what it says, it drops packetsdead and will not carry out any further processing. A packet that matches arule perfectly and is then Dropped will be blocked. Note that this actionmight in certain cases have an unwanted effect, since it could leave deadsockets around on either host. A better solution in cases where this is likelywould be to use the REJECT target, especially when you wantto block port scanners from getting too much information, such on as filteredports and so on. Also note that if a packet has the DROPaction taken on it in a subchain, the packet will not be processed in any ofthe main chains either in the present or in any other table. The packet is inother words totally dead. As we've seen previously, the target will not sendany kind of information in either direction, nor to intermediaries such asrouters.

LOG target

The LOG target is specially designed for logging detailedinformation about packets. These could for example be considered as illegal.Or, logging can be used purely for bug hunting and error finding. TheLOG target will return specific information on packets,such as most of the IP headers and other information considered interesting.It does this via the kernel logging facility, normallysyslogd. This information may then be read directly withdmesg, or from the syslogd logs, or withother programs or applications. This is an excellent target to use in debugyour rule-sets, so that you can see what packets go where and what rules areapplied on what packets. Note as well that it could be a really great idea touse the LOG target instead of the DROPtarget while you are testing a rule you are not 100% sure about on aproduction firewall, since a syntax error in the rule-sets could otherwisecause severe connectivity problems for your users. Also note that theULOG target may be interesting if you are using reallyextensive logging, since the ULOG target has support directlogging to MySQL databases and suchlike.

Note

Note that if you get undesired logging direct to consoles, this is notan iptables or Netfilterproblem, but rather a problem caused by your syslogd configuration - mostprobably /etc/syslog.conf. Read more in mansyslog.conf for information about this kind of problem.

The LOG target currently takes five options that could beof interest if you have specific information needs, or want to set differentoptions to specific values. They are all listed below.

TableÂ 6.17.Â LOG target options

Option--log-levelExampleiptables -A FORWARD -p tcp -j LOG --log-leveldebug ExplanationThis is the option to tell iptables andsyslog which log level to use. For a complete list of loglevels read the syslog.conf manual. Normally there arethe following log levels, or priorities as they are normally referred to:debug, info,notice, warning,warn, err,error, crit,alert, emerg andpanic. The keyword error isthe same as err, warn is thesame as warning and panic isthe same as emerg. Note that all three of these aredeprecated, in other words do not use error,warn and panic. The prioritydefines the severity of the message being logged. All messages are loggedthrough the kernel facility. In other words, setting kern.=info/var/log/iptables in your syslog.conf file andthen letting all your LOG messages iniptables use log level info, would make all messages appearin the /var/log/iptables file. Note that there may beother messages here as well from other parts of the kernel that uses the infopriority. For more information on logging I recommend you to read thesyslog and syslog.conf man-pages aswell as other HOWTOs etc.   Option--log-prefixExampleiptables -A INPUT -p tcp -j LOG --log-prefix "INPUTpackets"ExplanationThis option tells iptables to prefix all logmessages with a specific prefix, which can the easily be combined withgrep or other tools to track specific problems and outputfrom different rules. The prefix may be up to 29 letters long, includingwhite-spaces and other special symbols.
Option--log-tcp-sequenceExampleiptables -A INPUT -p tcp -j LOG--log-tcp-sequenceExplanationThis option will log the TCP Sequencenumbers, together with the log message. The TCPSequence number are special numbers that identify each packet andwhere it fits into a TCP sequence, as well as how thestream should be reassembled. Note that this option constitutes a securityrisk if the logs are readable by unauthorized users, or by the world for thatmatter. As does any log that contains output from iptables.   Option--log-tcp-optionsExampleiptables -A FORWARD -p tcp -j LOG--log-tcp-optionsExplanationThe --log-tcp-options option logs thedifferent options from the TCP packet headers and canbe valuable when trying to debug what could go wrong, or what has actuallygone wrong. This option does not take any variable fields or anything likethat, just as most of the LOG options don't.   Option--log-ip-optionsExampleiptables -A FORWARD -p tcp -j LOG--log-ip-optionsExplanationThe --log-ip-options option will log most ofthe IP packet header options. This works exactly the same as the--log-tcp-options option, but instead works on the IPoptions. These logging messages may be valuable when trying to debug or trackspecific culprits, as well as for debugging - in just the same way as theprevious option.

MARK target

The MARK target is used to set Netfiltermark values that are associated with specific packets. This target is onlyvalidin the mangle table, and will not work outside there.The MARK values may be used in conjunction with theadvancedrouting capabilities in Linux to send different packets through differentroutesand to tell them to use different queue disciplines (qdisc), etc. For moreinformation on advanced routing, check out the Linux Advanced Routing and Traffic Control HOW-TO. Note that the mark value isnot set within the actual package, but is an value that is associated withinthe kernel with the packet. In other words, you can not set aMARK for a packet and then expect theMARK still to be there on another host. If this is what youwant, you will be better off with the TOS target which willmangle the TOS value in the IP header.

TableÂ 6.18.Â MARK target options
Option--set-markExampleiptables -t mangle -A PREROUTING -p tcp --dport 22 -jMARK --set-mark 2ExplanationThe --set-mark option is required to set amark. The --set-mark match takes an integer value. Forexample, we may set mark 2 on a specific stream of packets, or on all packetsfrom a specific host and then do advanced routing on that host, to decrease orincrease the network bandwidth, etc.

MASQUERADE target

The MASQUERADE target is used basically the same as theSNAT target, but it does not require any--to-source option. The reason for this is that theMASQUERADE target was made to work with, for example,dial-up connections, or DHCP connections, which getsdynamic IP addresses when connecting to the network in question. This meansthat you should only use the MASQUERADE target withdynamically assigned IP connections, which we don't know the actual address ofat all times. If you have a static IP connection, you should instead use theSNAT target.

When you masquerade a connection, it means that we set the IP address usedon a specific network interface instead of the --to-sourceoption, and the IP address is automatically grabbed from the information aboutthe specific interface. The MASQUERADE target also has theeffect that connections are forgotten when an interface goes down, which isextremely good if we, for example, kill a specific interface. If we would haveused the SNAT target, we may have been left with a lot ofold connection tracking data, which would be lying around for days, swallowingup worth-full connection tracking memory. This is in general the correctbehavior when dealing with dial-up lines that are probable to be assigned adifferent IP every time it is brought up. In case we are assigned a differentIP, the connection is lost anyways, and it is more or less idiotic to keep theentry around.

It is still possible to use the MASQUERADE target insteadof SNAT even though you do have an static IP, however, itis not favorable since it will add extra overhead, and there may beinconsistencies in the future which will thwart your existing scripts andrender them "unusable".

Note that the MASQUERADE target is only valid within thePOSTROUTING chain in the nat table, just as theSNAT target. The MASQUERADE target takesone option specified below, which is optional.

TableÂ 6.19.Â MASQUERADE target

Option--to-portsExampleiptables -t nat -A POSTROUTING -p TCP -j MASQUERADE--to-ports 1024-31000ExplanationThe --to-ports option is used to setthe source port or ports to use on outgoing packets. Either you can specify asingle port like --to-ports 1025 or you may specify a portrange as --to-ports 1024-3000. In other words, the lowerport range delimiter and the upper port range delimiter separated with ahyphen. This alters the default SNAT port-selection as described in the SNAT target section. The--to-ports option is only valid if the rule match sectionspecifies the TCP or UDP protocols with the --protocolmatch.

MIRROR target

The MIRROR target is an experimental and demonstrationtarget only, and you are warned against using it, since it may result inreally bad loops hence, among other things, resulting in serious Denial ofService. The MIRROR target is used to invert the source anddestination fields in the IP header, and then toretransmit the packet. This can cause some really funny effects, and I'll betthat thanks to this target not just one red faced cracker has cracked his ownbox by now. The effect of using this target is stark, to say the least. Let'ssay we set up a MIRROR target for port 80 at computer A. Ifhost B were to come from yahoo.com, and try to access theHTTP server at host A, the MIRRORtarget would return the yahoo host's own web page (since this is where it camefrom).

Note that the MIRROR target is only valid within theINPUT, FORWARD andPREROUTING chains, and any user-defined chains whichare called from those chains. Also note that outgoing packets resulting fromthe MIRROR target are not seen by any of the normal chainsin the filter, nat or mangle tables, which could give rise to loops and otherproblems. This could make the target the cause of unforeseen headaches. Forexample, a host might send a spoofed packet to another host that uses theMIRROR command with a TTL of 255, at thesame time spoofing its own packet, so as to seem as if it comes from a thirdhost that uses the MIRROR command. The packet will thenbounce back and forth incessantly, for the number of hops there are to becompleted. If there is only 1 hop, the packet will jump back and forth 240-255times. Not bad for a cracker, in other words, to send 1500 bytes of data andeat up 380 kbyte of your connection. Note that this is a best case scenariofor the cracker or script kiddie, whatever we want to call them.

QUEUE target

The QUEUE target is used to queue packets to User-landprograms and applications. It is used in conjunction with programs orutilities that are extraneous to iptables and may be used, for example, withnetwork accounting, or for specific and advanced applications which proxy orfilter packets. We will not discuss this target in depth, since the coding ofsuch applications is out of the scope of this tutorial. First of all it wouldsimply take too much time, and secondly such documentation does not haveanything to do with the programming side of Netfilter and iptables. All ofthis should be fairly well covered in the Netfilter Hacking HOW-TO.

REDIRECT target

The REDIRECT target is used to redirect packets andstreams to the machine itself. This means that we could for exampleREDIRECT all packets destined for theHTTP ports to an HTTP proxylike squid, on our own host. Locally generated packets are mapped to the127.0.0.1 address. In other words, this rewrites the destination address toourown host for packets that are forwarded, or something alike. TheREDIRECT target is extremely good to use when we want, forexample, transparent proxying, where the LAN hosts do notknow about the proxy at all.

Note that the REDIRECT target is only valid within thePREROUTING and OUTPUT chainsof the nat table. It is also valid within user-defined chains that are onlycalled from those chains, and nowhere else. The REDIRECTtarget takes only one option, as described below.

TableÂ 6.20.Â REDIRECT target
Option--to-portsExampleiptables -t nat -A PREROUTING -p tcp --dport 80 -jREDIRECT --to-ports 8080ExplanationThe --to-ports option specifies thedestination port, or port range, to use. Without the--to-ports option, the destination port is never altered.This is specified, as above, --to-ports 8080 in case weonlywant to specify one port. If we would want to specify an port range, we woulddoit like --to-ports 8080-8090, which tells theREDIRECT target to redirect the packets to the ports 8080through 8090. Note that this option is only available in rules specifying theTCP or UDP protocol with the--protocol matcher, since it wouldn't make any senseanywhereelse.

REJECT target

The REJECT target works basically the same as theDROPtarget, but it also sends back an error message to the host sending the packetthat was blocked. The REJECT target is as of today onlyvalid in theINPUT, FORWARD andOUTPUT chains or their sub chains. After all, thesewould be the only chains in which it would make any sense to put this target.Note that all chains that use the REJECT target may only becalled by the INPUT,FORWARD, and OUTPUT chains,else they won't work. There is currently only one option which controls thenature of how this target works, though this may in turn take a huge set ofvariables. Most of them are fairly easy to understand, if you have a basicknowledge of TCP/IP.

TableÂ 6.21.Â REJECT target
Option--reject-withExampleiptables -A FORWARD -p TCP --dport 22 -j REJECT--reject-withtcp-resetExplanationThis option tells the REJECT target whatresponse to send to the host that sent the packet that we are rejecting. Oncewe get a packet that matches a rule in which we have specified this target,our host will first of all send the associated reply, and the packet will thenbe dropped dead, just as the DROP target would drop it. Thefollowing reject types are currently

RETURN target

The RETURN target will cause the current packet to stoptraveling through the chain where it hit the rule. If it is the subchain ofanother chain, the packet will continue to travel through the superior chainsas if nothing had happened. If the chain is the main chain, for example theINPUT chain, the packet will have the default policytaken on it. The default policy is normally set to ACCEPT,DROP or similar.

For example, let's say a packet enters the INPUT chain and then hits a rulethat it matches and that tells it to --jump EXAMPLE_CHAIN.The packet will then start traversing the EXAMPLE_CHAIN,and all of a sudden it matches a specific rule which has the --jumpRETURN target set. It will then jump back to theINPUT chain. Another example would be if the packethit a --jump RETURN rule in theINPUT chain. It would then be dropped to the defaultpolicy as previously described, and no more actions would be taken in thischain.

SNAT target

The SNAT target is used to do Source Network AddressTranslation, which means that this target will rewrite the Source IP addressinthe IP header of the packet. This is what we want, for example, when severalhosts have to share an Internet connection. We can then turn on ip forwardingin the kernel, and write an SNAT rule which will translateall packets going out from our local network to the sourceIP of our own Internet connection. Without doing this, the outsideworld would not know where to send reply packets, since our local networksmostly use the IANA specified IP addresses which are allocated forLAN networks. If we forwarded these packets as is, no oneon the Internet would know that they where actually from us. TheSNAT target does all the translation needed to do this kindof work, letting all packets leaving our LAN look as ifthey came from a single host, which would be our firewall.

The SNAT target is only valid within the nat table,within the POSTROUTING chain. This is in other wordsthe only chain in which you may use SNAT. Only the firstpacket in a connection is mangled by SNAT, and after thatall future packets using the same connection will also beSNATted. Furthermore, the initial rules in thePOSTROUTING chain will be applied to all the packetsin the same stream.

TableÂ 6.22.Â SNAT target
Option--to-sourceExampleiptables -t nat -A POSTROUTING -p tcp -o eth0 -j SNAT --to-source 194.236.50.155-194.236.50.160:1024-32000 ExplanationThe --to-source option is used tospecify which source the packet should use. This option, at its simplest,takes one IP address which we want to use for the sourceIP address in the IP header. If we want tobalance between several IP addresses, we can use a range of IP addresses,separated by a hyphen. The --to--source IP numbers could then, forinstance, be something like in the above example:194.236.50.155-194.236.50.160. The source IP for each stream that we openwould then be allocated randomly from these, and a single stream wouldalways use the same IP address for all packets within that stream. We canalso specify a range of ports to be used by SNAT. Allthe source ports would then be confined to the ports specified. The portbit of the rule would then look like in the example above, :1024-32000.This is only valid if -p tcp or -p udp was specified somewhere in thematch of the rule in question. iptables will always try to avoid makingany port alterations if possible, but if two hosts try to use the sameports, iptables will map one of them to another port. If no port range isspecified, then if they're needed, all source ports below 512 will bemapped to other ports below 512. Those between source ports 512 and 1023will be mapped to ports below 1024. All other ports will be mapped to 1024or above. As previously stated, iptables will always try to maintain thesource ports used by the actual workstation making the connection. Notethat this has nothing to do with destination ports, so if a client triesto make contact with an HTTP server outside thefirewall, it will not be mapped to the FTP controlport.

TOS target

The TOS target is used to set the Type ofService field within the IP header. The TOSfield consists of 8 bits which are used to help in routingpackets. This is one of the fields that can be used directly withiniproute2 and its subsystem for routing policies. Worthnoting, is that that if you handle several separate firewalls and routers,this is the only way to propagate routing information within the actual packetbetween these routers and firewalls. As previously noted, theMARK target - which sets a MARKassociated with a specific packet - is only available within the kernel, andcan not be propagated with the packet. If you feel a need to propagate routinginformation for a specific packet or stream, you should therefore set theTOS field, which was developed for this.

There are currently a lot of routers on the Internet which do a pretty bad jobat this, so as of now it may prove to be a bit useless to attemptTOS mangling before sending the packets on to theInternet. At best the routers will not pay any attention to theTOS field. At worst, they will look at theTOS field and do the wrong thing. However, as statedabove, the TOS field can most definitely be put togood use if you have a large WAN orLAN with multiple routers. You then in fact have thepossibility of giving packets different routes and preferences, based on theirTOS value - even though this might be confined toyour own network.

Caution

The TOS target is only capable of setting specific values,or named values on packets. These predefined TOSvalues can be found in the kernel include files, or more precisely, theLinux/ip.h file. The reasons are many, and you shouldactually never need to set any other values; however, there are ways aroundthis limitation. To get around the limitation of only being able to set thenamed values on packets, you can use the FTOS patch available at the Paksecured Linux Kernel patches site maintained by MatthewG. Marsh. However, be cautious with this patch! You should not need to use anyother than the default values, except in extreme cases.

Note

Note that this target is only valid within themangle table and can not be used outside it.

Note

Also note that some old versions (1.2.2 or below) of iptables provided abroken implementation of this target which did not fix the packet checksumupon mangling, hence rendered the packets bad and in need of retransmission.That in turn would most probably lead to further mangling and the connectionnever working.

The TOS target only takes one option as describedbelow.

TableÂ 6.23.Â TOS target

Option--set-tosExampleiptables -t mangle -A PREROUTING -p TCP --dport 22 -jTOS --set-tos 0x10ExplanationThe --set-tos option tells theTOS mangler what TOS value to setonpackets that are matched. The option takes a numeric value, either in hex orindecimal value. As the TOS value consists of 8 bits,thevalue may be 0-255, or in hex 0x00-0xFF. Note that in the standard TOS targetyou are limited to using the named values available (which should be more orless standardized), as mentioned in the previous warning. These values areMinimize-Delay (decimal value 16, hex value0x10), Maximize-Throughput (decimal value 8,hex value 0x08), Maximize-Reliability(decimal value 4, hex value 0x04),Minimize-Cost (decimal value 2, hex 0x02) orNormal-Service (decimal value 0, hex value0x00). The default value on most packets isNormal-Service, or 0. Note that you can, ofcourse, use the actual names instead of the actual hex values to set theTOS value; in fact this is generally to berecommended, since the values associated with the names may be changed infuture. For

TTL target

Caution

This patch requires the TTL patch from the patch-o-matic tree available im the base directory from http://www.netfilter.org/documentation/index.html#FAQ- The official Netfilter Frequently AskedQuestions. Also a good place to start at when wondering whatiptables and Netfilter is about. .

The TTL target is used to modify the Time To Live field in the IP header. One useful application of this is tochange all Time To Live values to the same value onall outgoing packets. One reason for doing this is if you have a bullyISP which don't allow you to have more than one machineconnected to the same Internet connection, and who actively pursue this. Setting all TTL values to the same value, will effectively make it a little bit harder for them to notify that you are doing this. We may then reset the TTL value for all outgoing packets to a standardized value, such as 64 as specified in Linux kernel.

For more information on how to set the default value used inLinux, read the ip-sysctl.txt, which you may find within the Other resources and links appendix.

The TTL target is only valid within themangle table, and nowhere else. It takes 3 options asof writing this, all of them described below in the table.

Table 6.24. TTL target

Option--ttl-setExampleiptables -t mangle -A PREROUTING -i eth0 -j TTL--ttl-set 64ExplanationThe --ttl-set option tells theTTL target which TTL value to seton the packet in question. A good value would be around 64 somewhere. It's nottoo long, and it is not too short. Do not set this value too high, since it mayaffect your network and it is a bit immoral to set this value to high, since thepacket may start bouncing back and forth between two mis-configured routers, andthe higher the TTL, the more bandwidth will be eatenunnecessary in such a case. This target could be used to limit how far away our clients are. A good case of this could be DNS servers, where we don't want the clients to be too far away. Option--ttl-decExampleiptables -t mangle -A PREROUTING -i eth0 -j TTL--ttl-dec 1ExplanationThe --ttl-dec option tells theTTL target to decrement the Time To Live value by theamount specified after the --ttl-dec option. In otherwords, if the TTL for an incoming packet was 53 and we had set--ttl-dec 3, the packet would leave our host with aTTL value of 49. The reason for this is that thenetworking code will automatically decrement the TTLvalue by 1, hence the packet will be decremented by 4 steps,from 53 to 49. This could for example be used when we want to limit how far away the people using our services are. For example, users should always use a close-by DNS, and hence we could match all packets leaving our DNS server and then decrease it by several steps. Of course, the--set-ttl may be a better idea for this usage. Option--ttl-incExampleiptables -t mangle -A PREROUTING -i eth0 -j TTL--ttl-inc 1ExplanationThe --ttl-inc option tells theTTL target to increment the Time To Livevalue with the value specified to the--ttl-incoption. This means that we should raise the TTL valuewith the value specified in the --ttl-inc option, and if wespecified --ttl-inc 4, a packet entering with aTTL of 53 would leave the host withTTL 56. Note that the same thing goes here, as for the previous example of the --ttl-dec option, where the network code will automatically decrement the TTL value by 1, which it always does. This may be used to make our firewall a bit more stealthy to trace-routes among other things. By setting the TTL one value higher for all incoming packets, we effectively make the firewall hidden from trace-routes. Trace-routes are a loved and hated thing, since they provide excellent information on problems with connections and where it happens, but at the same time, it gives the hacker/cracker some good information about your upstreams if they have targeted you. For a good example on how this could be used, see the Ttl-inc.txt script.

ULOG target

The ULOG target is used to provide user-space logging ofmatching packets. If a packet is matched and the ULOG targetis set, the packet information is multicasted together with the whole packetthrough a netlink socket. One or more user-space processes may then subscribe tovarious multicast groups and receive the packet. This is in other words a morecomplete and more sophisticated logging facility that is only used by iptablesand Netfilter so far, and it contains much better facilities for loggingpackets. This target enables us to log information toMySQL databases, and other databases, making it muchsimpler to search for specific packets, and to group log entries. Youcan find the ULOGD user-land applications at the ULOGD project page.

Table 6.25. ULOG target

Option--ulog-nlgroupExampleiptables -A INPUT -p TCP --dport 22 -jULOG --ulog-nlgroup 2ExplanationThe --ulog-nlgroup option tells theULOG target which netlink group to send the packet to. Thereare 32 netlink groups, which are simply specified as 1-32. If we would like toreach netlink group 5, we would simply write --ulog-nlgroup5. The default netlink group used is 1.   Option--ulog-prefixExampleiptables -A INPUT -p TCP --dport 22 -j ULOG--ulog-prefix "SSH connection attempt: "ExplanationThe --ulog-prefix option works just the sameas the prefix value for the standard LOG target. This optionprefixes all log entries with a user-specified log prefix. It can be 32characters long, and is definitely most useful to distinguish differentlog-messages and where they came from.

Option--ulog-cprangeExampleiptables -A INPUT -p TCP --dport 22 -j ULOG--ulog-cprange 100ExplanationThe --ulog-cprange option tells theULOG target how many bytes of the packet to send to theuser-space daemon of ULOG. If we specify 100 as above, wewould copy 100 bytes of the whole packet to user-space, which would include thewhole header hopefully, plus some leading data within the actual packet. If wespecify 0, the whole packet will be copied to user-space, regardless of thepackets size. The default value is 0, so the whole packet will be copied touser-space.   Option--ulog-qthresholdExampleiptables -A INPUT -p TCP --dport 22 -j ULOG--ulog-qthreshold 10ExplanationThe --ulog-qthreshold option tells theULOG target how many packets to queue inside the kernelbefore actually sending the data to user-space. For example, if we set thethreshold to 10 as above, the kernel would first accumulate 10 packets insidethe kernel, and then transmit it outside to the user-space as one single netlinkmulti part message. The default value here is 1 because of backwardcompatibility, the user-space daemon did not know how to handle multi-partmessages previously.

Chapter 7. rc.firewall file

Table of Contents

This chapter will deal with an example firewall setup and how the script file could look. We have used one of the basic setups and dug deeperinto how it works and what we do in it. This should be used to get a basic ideaon how to solve different problems and what you may need to think about beforeactually putting your scripts into work. It could be used as is with somechanges to the variables, but is not suggested since it may not work perfectlytogether with your network setup. As long as you have a very basic setuphowever, it will very likely run quite smooth with just a few fixes to it.

Note

note that there might be more efficient ways of making the rule-set,however, the script has been written for readability so that everyone canunderstand it without having to know too much BASH scripting before readingthis

example rc.firewall

OK, so you have everything set up and are ready to check out an exampleconfiguration script. You should at least be if you have come this far. Thisexample rc.firewall.txt (also included in the Example scripts code-base appendix) is fairly large but not a lot of comments in it. Instead of looking for comments, I suggest you read through the script file to get a basic hum about how it looks, and then you return here to get the nitty gritty about the whole script.

explanation of rc.firewall

Configuration options

The first section you should note within the example rc.firewall.txt is the configuration section. This should always be changed since it contains the information that is vital to your actual configuration. For example, your IP address will always change, hence it is available here. The $INET_IP should always be a fully valid IP address, if you got one (if not, then you should probably look closer at the rc.DHCP.firewall.txt, however, read on since this script will introduce a lot of interesting stuff anyways). Also, the $INET_IFACE variable should point to the actual device used for your Internet connection. This could beeth0, eth1,ppp0, tr0, etc just toname a few possible device names.

This script does not contain any special configuration options for DHCP orPPPoE, hence these sections are empty. The same goes for all sections that areempty, they are however left there so you can spot the differences between thescripts in a more efficient way. If you need these parts, then you could alwayscreate a mix of the different scripts, or (hold yourself) create your own fromscratch.

The Local Area Network section contains most of theconfiguration options for your LAN, which arenecessary. For example, you need to specify the IPaddress of the physical interface connected to theLAN as well as the IP rangewhich the LAN uses and the interface that the box isconnected to the LAN through.

Also, as you may see there is a Localhost configuration section. We doprovide it, however you will with 99% certainty not change any of the valueswithin this section since you will almost always use the 127.0.0.1IP address and the interface will almost certainly benamed lo. Also, just below the Localhost configuration,you will find a brief section that pertains to the iptables. Mainly, thissection only consists of the $IPTABLES variable, which willpoint the script to the exact location of the iptablesapplication. This may vary a bit, and the default location when compiling theiptables package by hand is /usr/local/sbin/iptables.However, many distributions put the actual application in another location suchas /usr/sbin/iptables and

Initial loading of extra modules

First, we see to it that the module dependencies files are upto date by issuing an /sbin/depmod -a command. Afterthis we load the modules that we will require for this script. Always avoidloading modules that you do not need, and if possible try to avoid havingmodules lying around at all unless you will be using them. This is for securityreasons, since it will take some extra effort to make additional rules thisway. Now, for example, if you want to have support for theLOG, REJECT andMASQUERADE targets and don't have this compiled staticallyinto your kernel, we load these modules as follows:

```
/sbin/insmod ipt_LOG
/sbin/insmod ipt_REJECT
/sbin/insmod ipt_MASQUERADE
```

Caution

In these scripts we forcedly load the modules, which could lead tofailures of loading the modules. If a module fails to load, it could dependupon a lot of factors, and it will generate an error message. If some of themore basic modules fail to load, its biggest probable error is that the module,or functionality, is statically compiled into the kernel. For furtherinformation on this subject, read the Problems loading modules section in the Common problems and questions appendix.

Next is the option to load ipt_ownermodule, which could for example be used to only allow certain users to makecertain connections, etc. I will not use that module in this example butbasically, you could allow only root to doFTP and HTTP connections toredhat.com and DROP all the others. You could also disallowall users but your own user and root to connect from your box to the Internet,might be boring for others, but you will be a bit more secure to bouncinghacker attacks and attacks where the hacker will only use your host as anintermediate host. For more information about theipt_owner match, look at the ??? section within the???chapter.

We may also load extra modules for the state matching code here. All modulesthat extend the state matching code and connection tracking code are calledip_conntrack_* and ip_nat_*.Connection tracking helpers are special modules that tells the kernel how toproperly track the specific connections. Without these so called helpers, thekernel would not know what to look for when it tries to track specificconnections. The NAT helpers on the other hand, areextensions of the connection tracking helpers that tells the kernel what to lookfor in specific packets and how to translate these so the connections willactually work. For example, FTP is a complex protocolby definition, and it sends connection information within the actual payload ofthe packet. So, if one of your NATed boxes connect toa FTP server on the Internet, it will send its ownlocal network IP address within the payload of thepacket, and tells the FTP server to connect to thatIP address. Since this local network address is notvalid outside your own network, the FTP server will notknow what to do with it and hence the connection will break down. TheFTP NAT helpers do all of the translations within theseconnections so the FTP server will actually know whereto connect. The same thing applies for DCC filetransfers (sends) and chats. Creating these kind of connections requires theIP address and ports to be sent within theIRC protocol, which in turn requires some translationto be done. Without these helpers, some FTP andIRC stuff will work no doubt, however, some otherthings will not work. For example, you may be able to receive files overDCC, but not be able to send files. This is due to howthe DCC starts a connection. First off, you tell thereceiver that you want to send a file and where he should connect to

Note

If you are experiencing problems with mIRC DCCs over your firewalland everything works properly with other IRC clients, read the mIRC DCC problems section in the Common problems and questions appendix.

As of this writing, there is only the option to load modules which add supportfor the FTP and IRCprotocols. For a long explanation of these conntrack and nat modules, read theCommon problems and questionsappendix. There are also H.323 conntrack helpers withinthe patch-o-matic, as well as some other conntrack aswell as NAT helpers. To be able to use these helpers,you need to use the patch-o-matic and compile your ownkernel. For a better explanation on how this is done, read the Preparations chapter.

Note

Note that you need to load the ip_nat_irc andip_nat_ftp if you want Network AddressTranslation to work properly on any of theFTP and IRC protocols. Youwill also need to load the ip_conntrack_irc andip_conntrack_ftp modules before actually loading theNAT modules. They are used the same way as theconntrack modules, but it will make it possible for the computer to doNAT on these two protocols.

proc set up

At this point we start the IP forwarding byechoing a 1 to /proc/sys/net/ipv4/ip_forward in thisfashion:

    echo "1" > /proc/sys/net/ipv4/ip_forward

Warning

It may be worth a thought where and when we turn on theIP forwarding. In this script and all otherswithin the tutorial, we turn it on before actually creating any kind ofIP filters (i.e., iptablesrule-sets). This will lead to a brief period of time where the firewallwill accept forwarding any kind of traffic for everything between amillisecond to a minute depending on what script we are running and onwhat box. This may give malicious people a small time-frame to actually getthrough our firewall. In other words, this option should really be turnedon after creating all firewall rules, however, I havechosen to turn it on before loading any rules to maintain consistency withthe script breakdown currently used in all scripts.

In case you need dynamic IP support, for example ifyou use SLIP, PPP orDHCP you may enable the next option,ip_dynaddr by doing the following :

    echo "1" > /proc/sys/net/ipv4/ip_dynaddr

If there is any other options you might need to turn on you should followthat style, there's other documentations on how to do these things and thisis out of the scope of this documentation. There is a good but rather briefdocument about the proc system available within the kernel, which is alsoavailable within the Other resources and links appendix. The Other resources and links appendix isgenerally a good place to start looking when you have specific areas that youare looking for information on, that you do not find here.

Note

The rc.firewall.txt script, and all other scriptscontained within this tutorial, do contain a small section of non-required procsettings. These may be a good primer to look at when something is not workingexactly as you want it to, however, do not change these values before actuallyknowing what they mean.

Displacement of rules to different chains

This section will briefly describe my choices within the tutorialregarding user specified chains and some choices specific to therc.firewall.txt script. Some of the paths I havechosen to go here may be wrong from one or another of aspect. I hope topoint these aspects and possible problems out to you when and where theyoccur. Also, this section contains a brief look back to the Traversing of tables and chainschapter. Hopefully, this will remind you a little bit of how the specifictables and chains are traversed in a real live example.

 I have displaced all the different user-chains in the fashion Ihave to save as much CPU as possible but at the same time put the mainweight on security and readability. Instead of letting aTCP packet traverseICMP, UDP andTCP rules, I simply match allTCP packets and then let theTCP packets traverse an user specified chain.This way we do not get too much overhead out of it all. The followingpicture will try to explain the basics of how an incoming packet traversesNetfilter. With these pictures and explanations, I wish to explain andclarify the goals of this script. We will not discuss any specific detailsyet, but instead further on in the chapter. This is a really trivialpicture in comparison to the one in the Traversing of tables and chains chapter where we discussed thewhole traversal of chains and tables in depth.

Based upon this picture, let us make clear what our goals are. This wholeexample script is based upon the assumption that we are looking at a scenariocontaining one local network, one firewall and an Internet connection connectedto the firewall. This example is also based upon the assumption that we have astatic IP to the Internet (as opposed toDHCP, PPP andSLIP and others). In this case, we also want to allowthe firewall to act as a server for certain services on the Internet, and wetrust our local network fully and hence we will not block any of the trafficfrom the local network. Also, this script has as a main priority to only allowtraffic that we explicitly want to allow. To do this, we want to set defaultpolicies within the chains to DROP. This willeffectively kill all connections and all packets that we do not explicitly allowinside our network or our firewall.

In the case of this scenario, we would also like to let our local network doconnections to the Internet. Since the local network is fully trusted, we wantto allow all kind of traffic from the local network to the Internet. However,the Internet is most definitely not a trusted network and hence we want toblock them from getting to our local network. Based upon these generalassumptions, let's look at what we need to do and what we do not need and wantto do.

First of all, we want the local network to be able to connect to theInternet, of course. To do this, we will need to NATall packets since none of the local computers have real IP addresses. All ofthis is done within the PREROUTING chain, which iscreated last in this script. This means that we will also have to do somefiltering within the FORWARD chain since we willotherwise allow outsiders full access to our local network. We trust our localnetwork to the fullest, and because of that we specifically allow all trafficfrom our local network to the Internet. Since no one on the Internet should beallowed to contact our local network computers, we will want to block alltraffic from the Internet to our local network except already established andrelated connections, which in turn will allow all return traffic from theInternet to our local network.

As for our firewall, we may be a bit low on funds perhaps, or we just wantto offer a few services to people on the Internet. Therefore, we have decidedto allow HTTP, FTP,SSH and IDENTD access to theactual firewall. All of these protocols are available on the actual firewall,and hence it should be allowed through the INPUT chain,and we need to allow the return traffic through theOUTPUT chain. However, we also trust the local networkfully, and the loopback device and IP address are alsotrusted. Because of this, we want to add special rules to allow all traffic fromthe local network as well as the loopback network interface. Also, we do notwant to allow specific packets or packet headers in specific conjunctions, nordo we want to allow some IP ranges to reach the firewall from the Internet. Forinstance, the 10.0.0.0/8 address range is reserved forlocal networks and hence we would normally not want to allow packets from such aaddress range since they would with 90% certainty be spoofed. However, before weimplement this, we must note that certain Internet Service Providers actuallyuse these address ranges within their own networks. For a closer discussion ofthis, read the Common problems and questions chapter.

Since we have an FTP server running on the server, aswell as the fact we want to traverse as few rules as possible, we add a rulewhich lets all established and related traffic through at the top of theINPUT chain. For the same reason, we want to split therules down into sub-chains. By doing this, our packets will hopefully only needto traverse as few rules as possible. By traversing less rules, we make therule-set less time consuming for each packet, and reduce redundancy within thenetwork.

In this script, we choose to split the different packets down by theirprotocol family, for example TCP,UDP or ICMP. AllTCP packets traverse a specific chain namedtcp_packets, which will contain rules for allTCP ports and protocols that we want to allow. Also, wewant to do some extra checking on the TCP packets, sowe would like to create one more subchain for all packets that are accepted forusing valid port numbers to the firewall. This chain we choose to call theallowed chain, and should contain a few extra checksbefore finally accepting the packet. As for ICMPpackets, these will traverse the icmp_packets chain.When we decided on how to create this chain, we could not see any specific needsfor extra checks before allowing the ICMP packetsthrough if we agree with the type and code of the ICMPpacket, and hence we accept them directly. Finally, we have the UDP packets whichneed to be dealt with. These packets, we send to theudp_packets chain which handles all incomingUDP packets. All incoming UDPpackets should be sent to this chain, and if they are of an allowed type weshould accept them immediately without any further checking.

Since we are running on a relatively small network, this box is also used as asecondary workstation and to give some extra levy for this, we want to allowcertain specific protocols to make contact with the firewall itself, such asspeak freely and ICQ.

Finally, we have the firewalls OUTPUT chain. Sincewe actually trust the firewall quite a lot, we allow pretty much all trafficleaving the firewall. We do not do any specific user blocking, nor do we do anyblocking of specific protocols. However, we do not want people to use this boxto spoof packets leaving the firewall itself, and hence we only want to allowtraffic from the IP addresses assigned to the firewall itself. We would mostlikely implement this by adding rules that ACCEPT allpackets leaving the firewall in case they come from one of the IPaddresses assigned to the firewall, and if not they will be droppedby the default policy in the OUTPUT chain.

Setting up default policies

Quite early on in the process of creating our rule-set, we set up the defaultpolicies. We set up the default policies on the different chains with a fairlysimple command, as described below.

iptables [-P {chain} {policy}]

The default policy is used every time the packets do not match a rule in thechain. For example, let's say we get a packet that match no single rule in ourwhole rule-set. If this happens, we must decide what should happen to the packetin question, and this is where the default policy comes into the picture. Thedefault policy is used on all packets that does not match with any other rule inour rule-set.

Caution

Do be cautious with what default policy you set on chains in othertables since they are simply not made for filtering, and it may lead to verystrange behaviors.

Setting up user specified chains in the filter table

Now you got a good picture on what we want to accomplish with this firewall, solet us get on to the actual implementation of the rule-set. It is now high timethat we take care of setting up all the rules and chains that we wish to createand to use, as well as all of the rule-sets within the chains.

After this, we create the different special chains that we want to usewith the -N command. The new chains are created and set upwith no rules inside of them. The chains we will use are, as previouslydescribed, icmp_packets,tcp_packets,udp_packets and theallowed chain, which is used by thetcp_packets chain. Incoming packets on$INET_IFACE, of ICMP type, will beredirected to the chain icmp_packets. Packets ofTCP type, will be redirected to thetcp_packets chain and incoming packets ofUDP type from $INET_IFACE go

toudp_packets chain. All of this will beexplained more in detail in the INPUT chain section below. To create a chain is quitesimple and only consists of a short declaration of the chain as this:

iptables [-N chain]

In the upcoming sections we will have a closer look at each and one of theuser defined chains that we have by now created. Let us have a closer look athow they look and what rules they contain and what we will accomplish withinthem.

The bad_tcp_packets chain

    The bad_tcp_packets chain is devoted to containrules that inspects incoming packets for malformed headers or other problems. Asit is, we have only chosen to include a packet filter which blocks all incomingTCP packets that are considered asNEW but does not have the SYN bitset, as well as a rule that blocks SYN/ACK packets that are considered NEW. This chain could be used to check for all possible inconsistencies, such as above or XMAS port-scans etc. We could also add rules that looks for state INVALID.

    If you want to fully understand the NEW not SYN, you need to look at theState NEW packets but no SYN bit set section in theCommon problems and questions appendixregarding state NEW and non-SYN packets getting through other rules. Thesepackets could be allowed under certain circumstances but in 99% of the cases wewouldn't want these packets to get through. Hence, we log them to our logs andthen we DROP them.

The reason that we REJECT SYN/ACK packets that are considered NEW is also very simple. It is described in more depth in the SYN/ACK and NEW packets section in theCommon problems and questions appendix.Basically, we do this out of courtesy to other hosts, since we will prevent themfrom being attacked in a sequence number prediction attack.

The allowed chain

If a packet comes in on $INET_IFACE and is ofTCP type, it travels through thetcp_packets chain and if the connection is against aport that we want to allow traffic on, we want to do some finalchecks on it to see if we actually do want to allow it or not. All of thesefinal checks are done within the allowed chain.

First of all, we check if the packet is a SYN packet.If it is a SYN packet, it is most likely to be thefirst packet in a new connection so, of course, we allow this. Then we check ifthe packet comes from an ESTABLISHED orRELATED connection, if it does, then we, again of course,allow it. An ESTABLISHED connection is a connection that hasseen traffic in both directions, and since we have seen aSYN packet, the connection then must be in stateESTABLISHED, according to the state machine. The last rule inthis chain will DROP everything else. In this case thispretty much means everything that has not seen traffic in both directions, i.e., we didn't reply to the SYN packet, or they aretrying to start the connection with a non SYN packet.There is no practical use of not starting a connectionwith a SYN packet, except to port scan people prettymuch. There is no currently availableTCP/IP implementation thatsupports opening a TCP connection with something elsethan a SYN packet to my knowledge, hence,DROP it since it is 99% sure to be a port scan.

The TCP chain

The tcp_packets chain specifies whatports that are allowed to use on the firewall from the Internet. Thereis, however, even more checks to do, hence we send each and one of the packetson to the allowed chain, which we described previously.

-A tcp_packets tells iptables in whichchain to add the new rule, the rule will be added to the end of the chain.-p TCP tells it to match TCP packetsand -s 0/0 matches all source addresses from 0.0.0.0 withnetmask 0.0.0.0, in other words all source addresses. Thisis actually the default behavior but I am using it just to make everything asclear as possible. --dport 21 means destination port21, in other words if the packet is destined for port 21 they also match. If allthe criteria are matched, then the packet will be targeted for theallowed chain. If it doesn't match any of the rules,they will be passed back to the original chain that sent the packet to thetcp_packets chain.

As it is now, I allow TCP port 21, orFTP control port, which is used to controlFTP connections and later on I also allow allRELATED connections, and that way we allow PASSIVE and ACTIVEconnections since the ip_conntrack_ftp module is,hopefully, loaded. If we do not want to allow FTP atall, we can unload the ip_conntrack_ftp module anddelete the $IPTABLES -A tcp_packets -p TCP -s 0/0 --dport21 -j allowed line from the rc.firewall.txt file.

Port 22 is SSH, which is much better than allowingtelnet on port 23 if you want to allow anyone from the outside to use a shell onyour box at all. Note that you are dealing with a firewall. It is always a badidea to give others than yourself any kind of access to a firewall box.Firewalls should always be kept to a bare minimum and no more.

Port 80 is HTTP, in other words your web server, deleteit if you do not want to run a web server directly on your firewall.

And finally we allow port 113, which is IDENTD andmight be necessary for some protocols like IRC, etc to work properly. Do notethat it may be worth to use the oidentd package if youNAT several hosts on your local network.oidentd has support for relayingIDENTD requests on to the correct boxes within yourlocal network.

If you feel like adding more open ports with this script, well, it shouldbe quite obvious how to do that by now. Just cut and paste one of the otherlines in the tcp_packets chain and change it to theport you want to open.

The UDP chain

If we do get a UDP packet on theINPUT chain, we send them on toudp_packets where we once again do a match forthe UDP protocol with -p UDP andthen match everything with a source address of 0.0.0.0 and netmask 0.0.0.0, inother words everything again. Except this, we only accept specificUDP ports that we want to be open for hosts on theInternet. Do note that we do not need to open up holes depending on the sendinghosts source port, since this should be taken care of by the state machine. Weonly need to open up ports on our host if we are to run a server on anyUDP port, such as DNS etc.Packets that are entering the firewall and that are part of an alreadyestablished connection (by our local network) will automatically be acceptedback in by the --state ESTABLISHED,RELATED rules atthe top of the INPUT chain.

As it is, we do not ACCEPT incomingUDP packets from port 53, which is what we use to doDNS lookups. The rule is there, but it is perdefault commented out. If you want your firewall to act as anDNS server, uncomment this line.

I personally also allow port 123, which is NTP ornetwork time protocol. This protocol is used to setyour computer clock to the same time as certain other time servers which havevery accurate clocks. Most of you probably do notuse this protocol and hence I am not allowing it per default. The same thingapplies here however, the rule is there and it is simple to uncomment to get itworking.

We do currently allow port 2074, which is used for certain real-timemultimedia applications like speakfreely which you can use to talk to other people in real-time by usingspeakers and a microphone, or even better, a headset. If you would not like touse this, you could turn it off quite simply by commenting it out.

Port 4000 is the ICQ protocol. This should be anextremely well known protocol that is used by the Mirabilis application namedICQ. There is at least 2-3 different ICQclones for Linux and it is one of the most widely used chat programs in theworld. I doubt there is any further need to explain what it is.

 At this point, two extra rules are available if you are experiencing a lotof log entries due to different circumstances. The first rule will blockbroadcast packets to destination ports 135 through 139. These are used byNetBIOS, or SMB for mostMicrosoft users. This will block all log entries we may get from MicrosoftNetworks on our outside otherwise. The second rule was also created totake care of excessive logging problems, but instead takes care ofDHCP queries from the outside. This isspecifically true if your outside network consists of a non-switchedEthernet type of network, where the clients receive their IPaddresses by DHCP. During thesecircumstances, you could wind up with a lot of logs from just that.

Note

Do note that the last two rules are specifically opted out since some people may be interested in these kind of logs. If you are experiencing problems with excessive legit logging, try to drop these types of packages at this point. There are also more rules of this type just before the log rules in the INPUT chain.

The ICMP chain

This is where we decide what ICMP types toallow. If a packet of ICMP type comes in oneth0 on the INPUT chain, we then redirect itto the icmp_packets chain as explainedbefore. Here we check what kind of ICMP typesto allow. For now, I only allow incoming ICMPEcho requests, TTL equals 0 duringtransit and TTL equals 0 duringreassembly. The reason that we do not allow any other ICMP typesper default here, is that almost all other ICMP types should be covered by theRELATED state rules.

Note

If an ICMP packet is sent as a reply to an already existing packet or packet stream it is considered RELATED to the original stream. For more information on the states, read the The state machine chapter.

The reason that I allow these ICMP packetsare as follows, Echo Requests are used to request an echo reply, which in turnis used to mainly ping other hosts to see if they are available on any of thenetworks. Without this rule, other hosts will not be able to ping us to see ifwe are available on any network connection. Do note that some people would tendto erase this rule, since they simple do not want to be seen on the Internet.Deleting this rule will effectively render any pings to our firewall totallyuseless from the Internet since the firewall will simply not respond to them.

Time Exceeded (i.e., TTL equals 0 during transit andTTL equals 0 during reassembly), is allowed in thecase we want to trace-route some host or if a packet gets its Time To Live setto 0, we will get a reply about this. For example, when you trace-routesomeone, you start out with TTL = 1, and it getsdown to 0 at the first hop on the way out, and a Time Exceeded is sent backfrom the first gateway en route to the host we are trying to trace-route, thenTTL = 2 and the second gateway sends Time Exceeded,and so on until we get an actual reply from the host we finally want to getto. This way, we will get a reply from each host on our way to the actual hostwe want to reach, and we can see every host in between and find out what hostis broken.

For a complete listing of all ICMP types, see theICMP types appendix . For moreinformation on ICMP types and their usage, i suggestreading the following documents and reports:

* The Internet Control Message Protocol by Ralph Walden.

* RFC 792 - Internet Control Message Protocol by J. Postel.

Note

As a side-note, I might be wrong in blocking some of theseICMP types for you, but in my case, everything worksperfectly while blocking all the ICMP types thatI do not allow.

INPUT chain

The INPUT chain as I have written it uses mostlyother chains to do the hard work. This way we do not get too much load fromiptables, and it will work much better on slow machines which might otherwisedrop packets at high loads. This is done by checking for specific details thatshould be the same for a lot of different packets, and then sending thosepackets into specific user specified chains. By doing this, we can split downour rule-set to contain much less rules that needs to be traversed by eachpacket and hence the firewall will be put through a lot less overhead by packetfiltering.

First of all we do certain checks for bad packets. This is done by sending allTCP packets to thebad_tcp_packets chain. This chain contains a few rulesthat will check for badly formed packets or other anomalies that we do not wantto accept. For a full explanation of the The bad_tcp_packets chain section in this chapter.

At this point we start looking for traffic from generally trusted networks.These include the local network adapter and all traffic coming fromthere, all traffic to and from our loopback interface,including all our currently assigned IP addresses (thismeans all of them, including our Internet IP address).As it is, we have chosen to put the rule that allowsLAN activity to the firewall at the top, since ourlocal network generates more traffic than the Internet connection. This allowsfor less overhead used to try and match each packet with each rule and it isalways a good idea to look through what kind of traffic mostly traverses thefirewall. By doing this, we can shuffle around the rules to be more efficient,leading to less overhead on the firewall and less congestion on your network.

Before we start touching the "real" rules which decides what we allow from theInternet interface and not, we have a related rule set up to reduce ouroverhead. This is a state rule which allows all packets part of an alreadyESTABLISHED or RELATED streamto the Internet IP address. This rule has an equivalentrule in the allowed chain, which are made rather redundant by this rule, whichwill be evaluated before the allowed ones are. However,the --state ESTABLISHED,RELATED rule in the allowed chain hasbeen retained for several reasons, such as people wanting to cut and pasting thefunction.

After this, We match all TCP packets in theINPUT chain that comes in on the$INET_IFACE interface, and send those to thetcp_packets, which was previously described.Now we do the same match for UDP packets on the$INET_IFACE and send those to theudp_packets chain, and after this allICMP packets are sent to theicmp_packets chain. Normally, a firewall would behardest hit by TCP packets, thenUDP and last of them all ICMPpackets. This is in normal case, mind you, and it may be wrong for you. Theabsolute same thing should be looked upon here, as with the network specificrules. Which causes the most traffic? Should the rules be thrown around togenerate less overhead? On networks sending huge amounts of data, thisis an absolute necessity since a Pentium III equivalent machine may be brought to its knees by a simple rule-set containing100 rules and a single 100mbit Ethernet card runningat full capacity if the rule-set is badly written. This is an important pieceto look at when writing a rule-set for your own local network.

At this point we have one extra rule, that is per default opted out, that can be used to get rid of some excessive logging in case we have some Microsoft network on the outside of our Linux firewall. Microsoft clients have a bad habit of sending out tons of multicast packets to the 224.0.0.0/8 range, and hence we have the opportunity to block those packets here so we don't fill our logs with them. There are also two more rules doing something similar tasks in the udp_packets chain described in the The UDP chain.

Before we hit the default policy of the INPUTchain, we log it so we may be able to find out about possible problems and/orbugs. Either it might be a packet that we just do not want to allow or it mightbe someone who is doing something bad to us, or finally it might be a problem inour firewall not allowing traffic that should be allowed. In either case we wantto know about it so it can be dealt with. Though, we do not log more than 3packets per minute as we do not want to flood our logs with crap which in turnmay fill up our whole logging partition, also we set a prefix to all log entriesso we know where it came from.

Everything that has not yet been caught will be DROPed bythe default policy on the INPUT chain. The defaultpolicy was set quite some time back, in the Setting up default policies section, in this chapter.

FORWARD chain

The FORWARD chain contains quite few rules in thisscenario. We have a single rule which sends all packets to thebad_tcp_packets chain, which was also used in theINPUT chain as described previously. Thebad_tcp_packets chain is constructed in such a fashionthat it can be used recycled in several calling chains, disregarding of whatpacket traverses it.

After this first check for bad TCP packets, wehave the main rules in the FORWARD chain. The firstrule will allow all traffic from our $LAN_IFACE to any otherinterface to flow freely, without restrictions. This rule will in other wordsallow all traffic from our LAN to the Internet. Thesecond rule will allow ESTABLISHED andRELATED traffic back through the firewall. This will inother words allow packets belonging to connections that was initiated from ourinternal network to flow freely back to our local network. These rules arerequired for our local network to be able to access the Internet, since thedefault policy of the FORWARD chain was previously setto DROP. This is quite clever, since it will allow hosts onour local network to connect to hosts on the Internet, but at the same timeblock hosts on the Internet from connecting to the hosts on our internal network.

Finally we also have a logging rule which will log packets that are notallowed in one or another way to pass through theFORWARD chain. This will most likely show one oranother occurrence of a badly formed packet or other problem. One cause may behacker attacks, and others may be malformed packets. This is exactly the samerule as the one used in the INPUT chain except for thelogging prefix, "IPT FORWARD packet died: ". The loggingprefix is mainly used to separate log entries, and may be used to distinguishlog entries to find out where the packet was logged from and some headeroptions.

OUTPUT chain

Since i know that there is pretty much no one but me using this box which ispartially used as a Firewall and a workstation currently, I allow almosteverything that goes out from it that has a source address$LOCALHOST_IP, $LAN_IP or$STATIC_IP. Everything else might be spoofed in some fashion,even though I doubt anyone that I know would do it on my box. Last of all we logeverything that gets dropped. If it does get dropped, we will mostdefinitely want to know about it so we may take action against the problem.Either it is a nasty error, or it is a weird packet that is spoofed. Finally weDROP the packet in the default policy.

PREROUTING chain of the nat table

The PREROUTING chain is pretty much what it says, itdoes network address translation on packets before they actually hit therouting decision that sends them onward to the INPUTor FORWARD chains in the filter table. The onlyreason that we talk about this chain in this script is that we once again feelobliged to point out that you should not do any filtering in it. ThePREROUTING chain is only traversed by the first packetin a stream, which means that all subsequent packets will go totally uncheckedin this chain. As it is with this script, we do not use thePREROUTING chain at all, however, this is the place wewould be working in right now if we wanted to do DNATon any specific packets, for example if you want to host your webserver within your local network. For more information about thePREROUTING chain, read the Traversing of tables and chains chapter.

Caution

The PREROUTING chain should not be used for anyfiltering since, among other things, this chain is only traversed by the firstpacket in a stream. The PREROUTING chain should be usedfor network address translation only, unless you really know what you aredoing.

Starting SNAT and the POSTROUTING chain

So, our final mission would be to get the NetworkAddress Translation up, correct? At least to me. First of allwe add a rule to the nat table, in thePOSTROUTING chain that willNAT all packets going out on our interfaceconnected to the Internet. For me this would beeth0. However, there are specific variables addedto all of the example scripts that may be used to automatically configurethese settings. The -t option tellsiptables which table to insert the rule in, in thiscase the nat table. The -A command tells us thatwe want to Append a new rule to an existing chain namedPOSTROUTING and -o $INET_IFACEtells us to match all outgoing packets on theINET_IFACE interface (or eth0,per default settings in this script) and finally we set the target toSNAT the packets. So all packets that match this rulewill be SNAT'ed to lookas it came from your Internet interface. Do note that you must set whichIP address to give outgoing packets with the--to-source option sent to the SNAT target.

In this script we have chosen to use the SNATtarget instead of MASQUERADE for a couple of reasons.The first one is that this script was supposed to run on a firewall thathas a static IP address. A follow up reason tothe first one, would hence be that it is faster and more efficient to usethe SNAT target if possible. Of course, it wasalso used to show how it would work and how it would be used in a reallive example. If you do not have a static IPaddress, you should definitely give thought to use theMASQUERADE target instead which provides a simple andeasy facility that will also do NAT for you, butthat will automatically grab the IP address thatit should use. This takes a little bit extra computing power, but it maymost definitely be worth it if you use DHCP forinstance. If you would like to have a closer look at how theMASQUERADE target may look, you should look at therc.DHCP.firewall.txt script.

ChapterÂ 8.Â Example scripts

Table of Contents

The objective of this chapter is to give a fairly brief and short explanation ofeach script available with this tutorial, and to provide an overlook of thescripts and what services they provide. These scripts are not in any wayperfect, and they may not fit your exact intentions perfectly. It is in otherwords up to you to make these scripts suitable for your needs. The rest of thistutorial should most probably be helpful in making this feat. The first sectionof this tutorial deals with the actual structure that I have established in eachscript so we may find our way within the script a bit easier.

rc.firewall.txt scriptstructure

 All scripts written for this tutorial has been written after a specificstructure. The reason for this is that they should be fairly conformable toeach other and to make it easier to find the differences between the scripts.This structure should be fairly well documented in this brief chapter. Thischapter should hopefully give a short understanding to why all the scripts hasbeen written as they have, and why I have chosen to maintain this structure.

Note

   Even though this is the structure I have chosen, do note that this may not bethe best structure for your scripts. It is only a structure that I have chosento use since it fits the need of being easy to read and follow the bestaccording to my logic.

The structure

   This is the structure that all scripts in this tutorial should follow. If theydiffer in some way it is probably an error on my part, unless it is specificallyexplained why I have broken this structure.

* Configuration - First of all we have theconfiguration options which the rest of the script should use. Configurationoptions should pretty much always be the first thing in any shell-script.

* Internet - This is the configuration sectionwhich pertains to the Internet connection. This could be skipped if we do nothave any Internet connection. Note that there may be more subsections than thoselisted here, but only such that pertains to our Internet connection.

* DHCP - If there are possibly any special DHCPrequirements with this specific script, we will add the DHCP specificconfiguration options here.

* PPPoE - If there are a possibility that theuser that wants to use this specific script, and if there are any specialcircumstances that raises the chances that he is using a PPPoE connection, wewill add specific options for those here.

* LAN - If there is any LAN available behind thefirewall, we will add options pertaining to that in this section. This is mostlikely, hence this section will almost always be available.

* DMZ - If there is any reason to it, we will adda DMZ zone configuration at this point. Most scripts lacks this section, mainlybecause any normal home network, or small corporate network, will not have one.

* Localhost - These options pertain to ourlocal-host. These variables are highly unlikely to change, but we have put mostof it into variables anyway. Hopefully, there should be no reason to changethese variables.

* iptables - This section contains iptablesspecific configuration. In most scripts and situations this should only requireone variable which tells us where the iptables binary is located.

* Other - If there are any other specificoptions and variables, they should first of all be fitted into the correctsubsection (If it pertains to the Internet connection, it should besub-sectioned there, etc). If it does not fit in anywhere, it should besub-sectioned directly to the configuration options somewhere.

* Module loading - This section of the scriptsshould maintain a list of modules. The first part should contain the requiredmodules, while the second part should contain the non-required modules.

Note

Note that some modules that may raise security, or add certain services orpossibilities, may have been added even though they are not required. Thisshould normally be noted in such cases within the example scripts.

* Required modules - This section should containthe required modules, and possibly special modules that adds to the security oradds special services to the administrator or clients.

* Non-required modules - This section containsmodules that are not required for normal operations. All of these modules shouldbe commented out per default, and if you want to add the service it provides, itis up to you.

* proc configuration - This section should takecare of any special configuration needed in the proc file system. If some ofthese options are required, they will be listed as such, if not, they should becommented out per default, and listed under the non-required procconfigurations. Most of the useful proc configurations will be listed here, butfar from all of them.

* Required proc configuration - This sectionshould contain all of the required proc configurations for the script inquestion to work. It could possibly also contain configurations that raisessecurity, and possibly which adds special services or possibilities for theadministrator or clients.

* Non-required proc configuration - This sectionshould contain non-required proc configurations that may prove useful. All ofthem should be commented out, since they are not actually necessary to get thescript to work. This list will contain far from all of the proc configurationsor nodes.

* rules set up - By now the scripts should mostprobably be ready to insert the rule-set. I have chosen to split all the rulesdown after table and then chain names. All user specified chains are createdbefore we do anything to the system built in chains. I have also chosen to setthe chains and their rule specifications in the same order as they are outputby the iptables -L command.

* Filter table - First of all we go through thefilter table and its content. First of all we should set up all the policies inthe table.

* Set policies - Set up all the default policiesfor the system chains. Normally I will set DROP policies on the chainsin the filter table, and specifically ACCEPT services and streams that I want toallow inside. This way we will get rid of all ports that we do not want to letpeople use.

* Create user specified chains - At this pointwe create all the user specified chains that we want to use later on within thistable. We will not be able to use these chains in the system chains anyways ifthey are not already created so we could as well get to it as soon as possible.

* Create content in user specified chains -After creating the user specified chains we may as well enter all the ruleswithin these chains. The only reason I have to enter this data at this pointalready is that may as well put it close to the creation of the user specifiedchains. You may as well put this later on in your script, it is totally up toyou.

* INPUT chain - When we have come this far, wedo not have a lot of things left to do within the filter table so we getonto the INPUT chain. At this point we should add all rules within the INPUTchain.

Note

At this point we start following the output from the iptables-L command as you may see. There is no reason for you to stay withthis structure, however, do try to avoid mixing up data from different tablesand chains since it will become much harder to read such rule-sets and to fixpossible problems.

* FORWARD chain - At this point we go on to addthe rules within the FORWARD chain. Nothing special about this decision.

* OUTPUT chain - Last of all in the filtertable, we add the rules dealing with the OUTPUT chain. There should hopefullynot be too much to do at this point.

* nat table - After the filter table we take careof the nat table. This is done after the filter table because of a number ofreasons within these scripts. First of all we do not want to turn the wholeforwarding mechanism and NAT function on at a too early stage, which couldpossibly lead to packets getting through the firewall at just the wrongtime point (i.e., when the NAT has been turned on, but none of the filterrules has been run). Also, I look upon the nat table as a sort of layer thatlies just outside the filter table and kind of surrounds it. The filter tablewould hence be the core, while the nat table acts as a layer lying around thefilter table, and finally the mangle table lies around the nat table as asecond layer. This may be wrong in some perspectives, but not too far fromreality.

* Set policies - First of all we set up all thedefault policies within the nat table. Normally, I will be satisfied with thedefault policy set from the beginning, namely the ACCEPT policy. This tableshould not be used for filtering anyways, and we should not let packets bedropped here since there are some really nasty things that may happen in suchcases due to our own presumptions. I let these chains be set to ACCEPT sincethere is no reason not to do so.

* Create user specified chains - At this pointwe create any user specified chains that we want within the nat table. NormallyI do not have any of these, but I have added this section anyways, just in case.Note that the user specified chains must be created before they can actually beused within the system chains.

* Create content in user specified chains - Bynow it should be time to add all the rules to the user specified chains in thenat table. The same thing goes here as for the user specified chains in thefilter table. We add this material here since I do not see any reason not to.

* PREROUTING chain - The PREROUTING chain isused to do DNAT on packets in case we have any need for it. In most scripts thisfeature is not used, or at the very least commented out, reason being that we donot want to open up big holes to our local network without knowing about it.Within some scripts we have this turned on by default since the sole purpose ofthose scripts are to provide such services.

* POSTROUTING chain - The POSTROUTING chainshould be fairly well used by the scripts I have written since most of themdepend upon the fact that you have one or more local networks that we want tofirewall against the Internet. Mainly we will try to use the SNAT target, but incertain cases we are forced to use the MASQUERADE target instead.

* OUTPUT chain - The OUTPUT chain is barely usedat all in any of the scripts. As it looks now, it is not broken, but I have beenunable to find any good reasons to use this chain so far. If anyone has a reasonto use this chain, send me a line and I will add it to the tutorial.

* mangle table - The last table to do anythingabout is the mangle table. Normally I will not use this table at all, since itshould normally not be used for anyone, unless they have specific needs, such asmasking all boxes to use the exact same TTL or to change TOS fields etc. Ihave in other words chosen to leave these parts of the scripts more or lessblank, with a few exceptions where I have added a few examples of what it may beused for.

* Set policies - Set the default policies withinthe chain. The same thing goes here as for the nat table pretty much. The tablewas not made for filtering, and hence you should avoid it all together. I havenot set any policies in any of the scripts in the mangle table one way or theother, and you are encouraged not to do so either.

* Create user specified chains - Create all theuser specified chains. Since I have barely used the mangle table at all in thescripts, I have neither created any chains here since it is fairly unusablewithout any data to use within it. However, this section was added just in casesomeone, or I, would have the need for it in the future.

* Create content in user specified chains - Ifyou have any user specified chains within this table, you may at this point addthe rules that you want within them here.

* PREROUTING - At this point there is barely anyinformation in any of the scripts in this tutorial that contains any rules here.

* INPUT chain - At this point there is barely anyinformation in any of the scripts in this tutorial that contains any rules here.

* FORWARD chain - At this point there is barely anyinformation in any of the scripts in this tutorial that contains any rules here.

* OUTPUT chain - At this point there is barely anyinformation in any of the scripts in this tutorial that contains any rules here.

* POSTROUTING chain - At this point there is barely anyinformation in any of the scripts in this tutorial that contains any rules here.

   Hopefully this should explain more in detail how each script is structured andwhy they are structured in such a way.

Caution

Do note that these descriptions are extremely brief, and should mainlyjust be seen as a brief explanation to what and why the scripts has been splitdown as they have. There is nothing that says that this is the only and best wayto go.

rc.firewall.txt

The rc.firewall.txt script is the main core on which the rest ofthe scripts are based upon. The rc.firewall file chapter should explain every detail inthe script most thoroughly. Mainly it was written for a dual homed network.For example, where you have one LAN and one InternetConnection. This script also makes the assumption that you have a static IP tothe Internet, and hence don't use DHCP,PPP, SLIP or some otherprotocol that assigns you an IP automatically. If you are looking for a scriptthat will work with those setups, please take a closer look at the rc.DHCP.firewall.txt script.

   The rc.firewall.txt script requires the followingoptions to be compiled statically to the kernel, or as modules. Without one ormore of these, the script will become more or less flawed since parts of thescripts required functionalities will be unusable. As you change the script youuse, you could possibly need more options to be compiled into your kerneldepending on what you want to use.

   * CONFIG_NETFILTER
   * CONFIG_IP_NF_CONNTRACK
   * CONFIG_IP_NF_IPTABLES
   * CONFIG_IP_NF_MATCH_LIMIT
   * CONFIG_IP_NF_MATCH_STATE
   * CONFIG_IP_NF_FILTER
   * CONFIG_IP_NF_NAT
   * CONFIG_IP_NF_TARGET_LOG

rc.DMZ.firewall.txt

The rc.DMZ.firewall.txt script was written for those people out therethat have one Trusted Internal Network, oneDe-Militarized Zone and one InternetConnection. The De-Militarized Zone isin this case 1-to-1 NATed and requires you to do someIP aliasing on your firewall, i.e., you must make the box recognize packetsfor more than one IP. There are several ways to get this to work, one is toset 1-to-1 NAT, another one if you have a wholesubnet is to create a subnetwork, giving the firewall one IP both internallyand externally. You could then set the IP's to theDMZed boxes as you wish. Do note that this will"steal" two IP's for you, one for the broadcast address and one for the networkaddress. This is pretty much up to you to decide and to implement, thistutorial will give you the tools to actually accomplish the firewalling andNATing part, but it will not tell you exactly whatyou need to do since it is out of the scope of the tutorial.

The rc.DMZ.firewall.txt script requires these options to be compiled intoyour kernel, either statically or as modules. Without these options, at the veryleast, available in your kernel, you will not be able to use this scriptsfunctionality. You may in other words get a lot of errors complaining aboutmodules and targets/jumps or matches missing. If you are planning to do trafficcontrol or any other things like that, you should see to it that you have allthe required options compiled into your kernel there as well.

* CONFIG_NETFILTER
* CONFIG_IP_NF_CONNTRACK
* CONFIG_IP_NF_IPTABLES
* CONFIG_IP_NF_MATCH_LIMIT
* CONFIG_IP_NF_MATCH_STATE
* CONFIG_IP_NF_FILTER
* CONFIG_IP_NF_NAT
* CONFIG_IP_NF_TARGET_LOG

You need to have two internal networks with this script as you can see from thepicture. One uses IP range 192.168.0.0/24 and consists of a TrustedInternal Network. The other one uses IP range 192.168.1.0/24 andconsists of the De-Militarized Zone which we will do1-to-1 NAT to. For example, if someone from theInternet sends a packet to our DNS_IP, then we useDNAT, to send the packet on to ourDNS on the DMZ network. Whenthe DNS sees our packet, the packet will be destinedfor the actual DNS internal network IP, and not to ourexternal DNS IP. If the packet would not have beentranslated, the DNS wouldn't have answered the packet.We will show a short example of how the DNAT codelooks:

```
$IPTABLES -t nat -A PREROUTING -p TCP -i $INET_IFACE -d $DNS_IP \
--dport 53 -j DNAT --to-destination $DMZ_DNS_IP
```

First of all, DNAT can only be performed in thePREROUTING chain of the nattable. Then we look for TCP protocol on our$INET_IFACE with destination IP that matches our$DNS_IP, and is directed to port 53, which is theTCP port for zone transfers between name servers. Ifwe actually get such a packet we give a target ofDNAT, in other words DNAT. After that we specify where we want the packet to go withthe --to-destination option and give a value of$DMZ_DNS_IP, in other words the IP of theDNS on our DMZ network. Thisis how basic DNAT works. When the reply to theDNATed packet is sent through the firewall, itautomatically gets un-DNATed.

By now you should have enough understanding of how everything works to be ableto understand this script pretty well without any huge complications. If thereis something you don't understand, that hasn't been gone through in the rest ofthe tutorial, mail me since it is probably a fault on my side.

rc.DHCP.firewall.txt

The rc.DHCP.firewall.txt script is pretty much identical to the originalrc.firewall.txt. However,this script no longer uses the STATIC_IP variable, which isthe main change to the original rc.firewall.txt script. The reason is thatthis won't work together with a dynamic IP connection. The actual changesneeded to be done to the original script is minimal, however, I've had somepeople mail me and ask about the problem so this script will be a goodsolution for you. This script will allow people who usesDHCP, PPP andSLIP connections to connect to the Internet.

The rc.DHCP.firewall.txt script requires the followingoptions to be compiled statically to the kernel, or as modules, as a bareminimum to run properly.

* CONFIG_NETFILTER
* CONFIG_IP_NF_CONNTRACK
* CONFIG_IP_NF_IPTABLES
* CONFIG_IP_NF_MATCH_LIMIT
* CONFIG_IP_NF_MATCH_STATE
* CONFIG_IP_NF_FILTER
* CONFIG_IP_NF_NAT
* CONFIG_IP_NF_TARGET_MASQUERADE
* CONFIG_IP_NF_TARGET_LOG

The main changes done to the script consists of erasing theSTATIC_IP variable as I already said and deleting allreferences to this variable. Instead of using this variable the script now does its main filtering on the variable INET_IFACE. Inother words -d $STATIC_IP has been changed to -i$INET_IFACE. This is pretty much the only changes made and that's allthat's needed really.

There are some more things to think about though. We can no longer filter in theINPUT chain depending on, for example,--in-interface $LAN_IFACE --dst $INET_IP. This in turn forcesus to filter only based on interfaces in such cases where the internal machinesmust access the Internet addressable IP. One great example is if we arerunning an HTTP on our firewall. If we go to the mainpage, which contains static links back to the same host, which could be somedyndns solution, we would get a real hard trouble. TheNATed box would ask the DNSfor the IP of the HTTP server, then try to accessthat IP. In case we filter based on interface and IP, theNATed box would be unable to get to theHTTP because the INPUT chainwould DROP the packets flat to the ground. This alsoapplies in a sense to the case where we got a static IP, but in such cases itcould be gotten around by adding rules which check theLAN interface packets for ourINET_IP, and if so ACCEPT them.

As you may read from above, it may be a good idea to get a script, orwrite one, that handles dynamic IP in a better sense. We could for examplemake a script that grabs the IP from ifconfig and adds it toa variable, upon boot-up of the Internet connection. A good way to do this, would be to use for example the ip-up scripts providedwith pppd and some other programs. For a good site, checkout the linuxguruz.org iptables site which has a huge collection of scriptsavailable to download. You will find a link to the linuxguruz.org site fromthe Other resources and linksappendix.

Note

This script might be a bit less secure than therc.firewall.txt script. I would definitely advise you touse that script if at all possible since this script is more open to attacksfrom the outside.

Also, there is the possibility to add something like this to your scripts:

```
INET_IP=`ifconfig $INET_IFACE | grep inet | cut -d : -f 2 | \
cut -d ' ' -f 1`
```

The above would automatically grab the IP addressof the $INET_IFACE variable, grep the correct line whichcontains the IP address and then cuts it downto a manageable IP address. For a moreelaborate way of doing this, you could apply the snippets of code availablewithin the retreiveip.txt script,which will automatically grab your Internet IP address when you run the script.Do note that this may in turn lead to a little bit of "weird" behaviors, suchas stalling connections to and from the firewall on the internal side. Themost common strange behaviors are described in the following list.

* If the script is run from within a script which in turn is executed by,for example, the PPP daemon, it will hang all currentlyactive connections due to the NEW not SYN rules (see the State NEW packets but no SYN bit set section). It is possibleto get by, if you get rid of the NEW not SYN rules forexample, but it is questionable.

* If you got rules that are static and always want to be around, it israther harsh to add and erase rules all the time, without hurting the alreadyexisting ones. For example, if you want to block hosts on yourLAN to connect to the firewall, but at the same timeoperate a script from the PPP daemon, how would you doit without erasing your already active rules blocking theLAN?

* It may get unnecessarily complicated, as seen above which in turn couldlead to security compromises. If the script is kept simple, it is easier to spotproblems, and to keep order in it.

rc.UTIN.firewall.txt

The rc.UTIN.firewall.txt script will in contrast to the other scripts block theLAN that is sitting behind us. In other words, we don'ttrust anyone on any networks we are connected to. We also disallow people on ourLAN to do anything but specific tasks on the Internet.The only things we actually allow is POP3,HTTP and FTP access to theInternet. We also don't trust the internal users to access the firewall morethan we trust users on the Internet.

The rc.UTIN.firewall.txt script requires the followingoptions to be compiled statically to the kernel, or as modules. Without one ormore of these, the script will become more or less flawed since parts of thescripts required functionalities will be unusable. As you change the script youuse, you could possibly need more options to be compiled into your kerneldepending on what you want to use.

*   CONFIG_NETFILTER
*   CONFIG_IP_NF_CONNTRACK
*   CONFIG_IP_NF_IPTABLES
*   CONFIG_IP_NF_MATCH_LIMIT
*   CONFIG_IP_NF_MATCH_STATE
*   CONFIG_IP_NF_FILTER
*   CONFIG_IP_NF_NAT
*   CONFIG_IP_NF_TARGET_LOG

This script follows the golden rule to not trust anyone, noteven our own employees. This is a sad fact, but a large part of the hacks andcracks that a company gets hit by is a matter of people from their own staffperpetrating the hit. This script will hopefully give you some clues as to whatyou can do with your firewall to strengthen it up. It's not very different fromthe original rc.firewall.txt script, but it does give a fewhints at what we would normally let through etc.

rc.test-iptables.txt

The rc.test-iptables.txt script can be used to test all the differentchains, but it might need some tweaking depending on your configuration, suchas turning on ip_forwarding, and setting upmasquerading etc. It will work for mostly everyonethough who has all the basic set up and all the basic tables loaded into kernel.All it really does is set some LOG targets which will logping replies and ping requests. This way, you will get information on whichchain was traversed and in which order. For example, run this script and thendo:

```
ping -c 1 host.on.the.internet
```

And tail -n 0 -f /var/log/messages while doing the firstcommand. This should show you all the different chains used and in which order,unless the log entries are swapped around for some reason.

Note

This script was written for testing purposes only. In other words,it's not a good idea to have rules like this that logs everything of one sortsince your log partitions might get filled up quickly and it would be aneffective Denial of Service attack against you and might lead to real attacks onyou that would be unlogged after the initial Denial of Service attack.

rc.flush-iptables.txt

The rc.flush-iptables.txt script should not really be calleda script in itself. The rc.flush-iptables.txt script will reset and flushall your tables and chains. The script starts by setting the default policiesto ACCEPT on the INPUT,OUTPUT and FORWARD chains ofthe filter table. After this we reset the defaultpolicies of the PREROUTING,POSTROUTING and OUTPUTchains of the nat table. We do this first so we won'thave to bother about closed connections and packets not getting through. Thisscript is intended for actually setting up and troubleshooting your firewall,and hence we only care about opening the whole thing up and reset it todefault values.

After this we flush all chains first in the filtertable and then in the NAT table. This way we know thereis no redundant rules lying around anywhere. When all of this is done, we jumpdown to the next section where we erase all the user specified chains in theNAT and filter tables. Whenthis step is done, we consider the script done. You may consider adding rulesto flush your mangle table if you use it.

Note

One final word on this issue. Certain people have mailed me asking from me toput this script into the original rc.firewall script using Red Hat Linuxsyntax where you type something like rc.firewall start and the script starts.However, I will not do that since this is a tutorial and should be used as aplace to fetch ideas mainly and it shouldn't be filled up with shell scriptsand strange syntax. Adding shell script syntax and other things makes thescript harder to read as far as I am concerned and the tutorial was writtenwith readability in mind and will continue being so.

Limit-match.txt

The limit-match.txt script is a minor test script which will let you test the limit match and see how it works. Load the script up, and then send ping packets at different intervals to see which gets through, and how often they get through. All echo replies will be blocked until the threshold for the burst limit has again been reached.

Pid-owner.txt

The pid-owner.txtis a small example script that shows how we could use the PID owner match. It does nothing real, but you should be able to run the script, and then from the output of iptables -L -v be able to tell that the rule actually matches.

Sid-owner.txt

The sid-owner.txt is a small example script that shows how we could use the SID ownermatch. It does nothing real, but you should be able to run the script, andthen from the output of iptables -L -v be able to tellthat the rule actually matches.

Ttl-inc.txt

A small example ttl-inc.txt script. This script shows how we could make the firewall/router invisible to traceroutes, which would otherwise reveal much information to possible attackers.

Iptables-save ruleset

A small example script used in the Saving and restoring large rule-sets chapter to illustrate how iptables-save may be used. This script is non-working, and should hence not be used for anything else than a reference.

AppendixÂ A.Â Detailed explanations of special commands

Table of Contents

Listing your active rule-set

To list your currently active rule-set you run a special option to theiptables command, which we have discussed briefly previouslyin the How a rule is built chapter. This would look like thefollowing:

    iptables -L

This command should list your currently active rule-set, and translateeverything possible to a more readable form. For example, it willtranslate all the different ports according to the/etc/services file as well as DNSall the IP addresses to get DNS records instead. Thelater can be a bit of a problem though. For example, it will try to resolveLAN IP addresses, i.e.192.168.1.1, to something useful.192.168.0.0/16 is a private range though andshould not resolve to anything and the command will seem to hang while resolvingthe IP. To get around this problem we would do something like the following:

    iptables -L -n

Another thing that might be interesting is to see a few statistics abouteach policy, rule and chain. We could get this by adding the verboseflag. It would then look something like this:

    iptables -L -n -v

Don't forget that it is also possible to list the nat and mangle tables. This is done with the -t switch, like this:

    iptables -L -t nat

There are also a few files that might be interesting to look at in the/proc file system. For example, it might be interestingto know what connections are currently in the conntrack table. This tablecontains all the different connections currently tracked and serves as a basictable so we always know what state a connection currently is in. This tablecan not be edited and even if it was possible, it would be a bad idea. Tosee the table you can run the following command:

cat /proc/net/ip_conntrack | less

The above command will show all currently tracked connections even thoughit might be a bit hard to understand everything.

Updating and flushing your tables

If at some point you screw up your iptables,there are actually commands to flush them, so you don't have to reboot.I've actually gotten this question a couple times by now so I thought I'danswer it right here. If you added a rule in error, you might just changethe -A parameter to -D in the lineyou added in error. iptables will find the erroneousline and erase it for you, in case you've got multiple lines lookingexactly the same in the chain, it erases the first instance it findsmatching your rule. If this is not the wanted behavior you might try touse the -D option as iptables -D INPUT10 which will erase the 10th rule in theINPUT chain.

There are also instances where you want to flush a whole chain,in this case you might want to run the -F option. Forexample, iptables -F INPUT will erase the wholeINPUT chain, though, this will not changethe default policy, so if this is set toDROP you'll block the wholeINPUT chain if used as above. To resetthe chain policy, do as how you set it toDROP, for example iptables -PINPUT ACCEPT.

I have made a rc.flush-iptables.txt (available as an appendix as well) that will flush and reset your iptables that you might consider using while setting up your rc.firewall.txt file properly. One thing though, if you start mucking around in the mangle table, this script will not erase those, it is rather simple to add the few lines needed to erase those but I have not added those here since the mangle table is not used in myrc.firewall.txt script so far.

Appendix B. Common problems and questions

Table of Contents
Problems loading modulesState NEW packets but no SYN bit setSYN/ACK and NEW packetsInternet Service Providers who use assigned IP addressesLetting DHCP requests through iptablesmIRC DCC problems

Problems loading modules

You may run into a few problems with loading modules. For example, you couldget errors claiming that there is no module by such a name and so on. This may,for example look like the following.

```
insmod: iptable_filter: no module by that name found
```

This is no reason for concern yet. This or these modules may possiblyhave been statically compiled into your kernel. This is the first thing youshould look at when trying to solve this problem. The simplest way to see ifthese modules have been loaded already or if they are statically compiled intothe kernel, is to simply try and run a command that uses the specificfunctionality. In the above case, we could not load thefilter table. If this functionality is not there, weshould be unable to use the filter table at all. Tocheck if the filter table is there, we do thefollowing.

```
iptables -t filter -L
```

This should either output all of the chains in thefilter table properly, or it should fail. If everythingis o.k., then it should look something like this depending on if you haverules inserted or not.

```
Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

If you do not have the filter table loaded, youwould get an error that looks something like this instead.

```
iptables v1.2.5: can't initialize iptables table `filter': Table \
    does not exist (do you need to insmod?)
Perhaps iptables or your kernel needs to be upgraded.
```

This is a bit more serious since it points out that we first of all do nothave the functionality compiled into the kernel, and second, that the module isnot possible to find in our normal module paths. This may either mean that youhave forgotten to install your modules, you have forgotten to rundepmod -a to update your module databases or you have notcompiled the functionality as either module or statically into kernel. There mayof course be other reasons for the module not to be loaded, but these are themain reasons. Most of these problems are easily solved. The first problem wouldsimply be solved by running make modules_install in thekernel source directory (if the source has already been compiled and the moduleshave already been built). The second problem is solved by simply runningdepmod -a once and see if it works afterward. The thirdproblem is a bit out of the league for this explanation, and you are more orless left to your own wits here. You will most probably find more informationabout this on the Linux Documentation Project homepage.

Another error that you may get when running iptables is the following error.

```
iptables: No chain/target/match by that name
```

This error tells us that there is no such chain, target or match. This coulddepend upon a huge set of factors, the most common being that you havemisspelled the chain, target or match in question. Also, this could begenerated in case you are trying to use a match that is not available, eitherbecause you did not load the proper module, it was not compiled into kernel oriptables failed to automatically load the module. In general, you should lookfor all of the above solutions but also look for misspelled targets of somesort or another in your rule.

State NEW packets but no SYN bit set

There is a certain feature iniptables that is not so well documented and may therefore beoverlooked by a lot of people (yes, including me). If you use stateNEW, packets with the SYN bit unsetwill get through your firewall. This feature is there because in certain caseswe want to consider that a packet may be part of an alreadyESTABLISHED connection on, for instance, another firewall.This feature makes it possible to have two or more firewalls, and for one of thefirewalls to go down without any loss of data. The firewalling of the subnetcould then be taken over by our secondary firewall. This does however lead tothe fact that state NEW will allow pretty much any kind ofTCP connection, regardless if this is the initial 3-way handshake or not. Totake care of this problem we add the following rules to our firewallsINPUT, OUTPUT andFORWARD chain:

```
$IPTABLES -A INPUT -p tcp ! --syn -m state --state NEW -j LOG \
     --log-prefix "New not syn:"
$IPTABLES -A INPUT -p tcp ! --syn -m state --state NEW -j DROP
```

Caution

The above rules will take care of this problem. This is a badly documentedbehavior of the Netfilter/iptablesproject and should definitely be more highlighted. In other words, a hugewarning is in its place for this kind of behavior on your firewall.

Note that there are some troubles with the above rules and bad Microsoft TCP/IPimplementations. The above rules will lead to certain conditions where packetsgenerated by Microsoft products gets labeled as state NEWand hence get logged and dropped. It will however not lead to brokenconnections to my knowledge. The problem occurs when a connection gets closed,the final FIN/ACK issent, the state machine of Netfilter closesthe connection and it is no longer in the conntrack table. At this point thefaulty Microsoft implementation sends another packet which is considered asstate NEW but lacks the SYN bitand hence gets matched by the above rules. In other words, don't worry to muchabout this rule, or if you are worried anyways, set the--log-headers option to the rule and log the headers tooand you'll get a better look at what the packet looks like.

There is one more known problem with these rules. If someone is currentlyconnected to the firewall, let's say from the LAN,and you have the script set to be activated when running aPPP connection. In this case, when you start thePPP connection, the person previously connected throughthe LAN will be more or less killed. This only applieswhen you are running with the conntrack and nat code bases as modules, and themodules are loaded and unloaded each time you run the script. Another way to getthis problem is to run the rc.firewall.txt script from atelnet connection from a host not on the actual firewall. To put it simple, youconnect with telnet or some other stream connection. Startthe connection tracking modules, then load the NEW notSYN packet rules. Finally, the telnetclient or daemon tries to send something. theconnection tracking code will not recognize this connection as a legalconnection since it has not seen packets in any direction on this connectionbefore, also there will be no SYN bits set since it isnot actually the first packet in the connection. Hence, the packet will match tothe rules and be logged and after-wards dropped to the ground.

SYN/ACK and NEW packets

Certain TCP spoofing attacks uses a technique called Sequence Number Prediction. In this type of attack, the attacker spoofs some other hosts IP address, while attacking someone, and tries to predict the Sequence number used by that host.

Let's look on typical TCP spoofing by sequence number prediction.Players: "attacker" [A], trying to send packets to a "victim" [V],pretending to be some "other host" [O].

* [A] sends SYN to [V] with source IP of [O].

* [V] replies to [O] by SYN/ACK.

* now [O] should reply to an unknown SYN/ACK by RST and the attack isunsuccesful, but let's assume [O] is down (flooded, turned off or behind firewall that has dropped the packets).

* if [O] didn't mess it up, [A] now can talk to [V] pretending to be[O] as long as it predicts correct sequence numbers.

As long as we do not send the RST packet to the unknown SYN/ACK in step 3, we will allow [V] to be attacked, and ourself to be incriminated. Common courtesy, would hence be to send the RST to [V] in a proper way. If we use the NEW not SYNrules specified in the ruleset, SYN/ACK packets will be dropped. Hence, we have the following rules in the bad_tcp_packets chain, just above the NEW not SYN rules:

```
iptables -A bad_tcp_packets -p tcp --tcp-flags SYN,ACK SYN,ACK \
-m state --state NEW -j REJECT --reject-with tcp-reset
```

The chance of being [O] in this scenario should be relatively small, but these rules should be safe in almost all cases. Except when you run several redundant firewalls which will often take over packets or streams from each other. In suchcase, some connections may be blocked, even though they are legit. This rule may actually also allow a few portscans to see our firewall as well, but they should not be able to tell much more than that.

Internet Service Providers who use assigned IP addresses

I have added this since a friend of mine told me something I havetotally forgotten. Certain stupid Internet Service Providers useIP addresses assigned by IANA for their localnetworks on which you connect to. For example, the Swedish Internet ServiceProvider and phone monopoly Telia uses this approach for example on theirDNS servers, which uses the 10.x.x.x IP address range.The problem you will most probably run into is that we, in this script, do notallow connections from any IP addresses in the 10.x.x.x range to us,because of spoofing possibilities. Well, here is unfortunately an examplewhere you actually might have to lift a bit on those rules. You might justinsert an ACCEPT rule above the spoofsection to allow traffic from those DNS servers, oryou could just comment out that part of the script. This is how it might look:

```
/usr/local/sbin/iptables -t nat -I PREROUTING -i eth1 -s \
      10.0.0.1/32 -j ACCEPT
```

 I would like to take my moment to bitch at these Internet ServiceProviders. These IP address ranges are not assigned for you to use for dumbstuff like this, at least not to my knowledge. For large corporate sites it ismore than o.k., or your own home network, but you are not supposed to force usto open up ourself just because of some winge of yours.

Letting DHCP requests through iptables

This is a fairly simple task really, once you get to know howDHCP works, however, you must be a little bit cautiouswith what you do let in and what you do not let in. First of all, we should knowthat DHCP works over the UDPprotocol. Hence, this is the first thing to look for. Second, we should checkwhich interface we get and send the request from. For example, if oureth0 interface is set up withDHCP, we should not allow DHCPrequests on eth1. To make the rule a bit more specific,we only allow the actual UDP ports used byDHCP, which should be ports 67 and 68. These are thecriteria that we choose to match packets on, and that we allow. The rule wouldnow look like this:

```
$IPTABLES  -I INPUT -i $LAN_IFACE -p udp --dport 67:68 --sport \
      67:68 -j ACCEPT
```

Do note that we allow all traffic to and from UDPport 67 and 68 now, however, this should not be such a huge problem since itonly allows requests from hosts doing the connection from port 67 or 68 as well.This rule could, of course, be even more restrictive, but it should be enough toactually accept all DHCP requests and updates withoutopening up too large holes. If you are concerned, this rule could of course bemade even more restrictive.

mIRC DCC problems

mIRC uses a special setting which allows it to connect through a firewall andto make DCC connections work properly without the firewall knowing about it. Ifthis option is used together with iptables and specifically the ip_conntrack_ircand ip_nat_irc modules, it will simply not work. The problem is that mIRC willautomatically NAT the inside of the packets for you, and when the packet reachesthe firewall, the firewall will simply not know how and what to do with it.mIRC does not expect the firewall to be smart enough to take care of this byitself by simply querying the IRC server for its IP address and sending DCCrequests with that address instead.

Turning on the "I am behind a firewall" configuration option and using theip_conntrack_irc and ip_nat_irc modules will result in Netfilter creating logentries with the following content "Forged DCC send packet".

The simplest possible solution is to just uncheck that configuration optionin mIRC and let iptables do the work. This means, that you should tell mIRCspecifically that it is not behind a firewall.

Appendix C. ICMP types

This is a complete listing of all ICMP types:

Table C.1. ICMP types

TYPECODEDescriptionQueryError00Echo Replyx 30Network Unreachable x31Host Unreachable x32Protocol Unreachable x33Port Unreachable x34Fragmentation needed but no frag. bit set x35Source routing failed x36Destination network unknown x37Destination host unknown x38Source host isolated (obsolete) x39Destination network administratively prohibited x310Destination host administratively prohibited x311Network unreachable for TOS x312Host unreachable for TOS x313Communication administratively prohibited by filtering x314Host precedence violation x315Precedence cutoff in effect x40Source quench 50Redirect for network 51Redirect for host 52Redirect for TOS and network 53Redirect for TOS and host 80Echo requestx 90Router advertisement 100Router solicitation 110TTL equals 0 during transit x111TTL equals 0 during reassembly x120IP header bad (catchall error) x121Required options missing x130Timestamp request (obsolete)x 14 Timestamp reply (obsolete)x 150Information request (obsolete)x 160Information reply (obsolete)x 170Address mask requestx 180Address mask replyx

Appendix D. Other resources and links

Here is a list of links to resources and where I have gotten informationfrom, etc :

* ip-sysctl.txt - from the 2.4.14 kernel. Alittle bit short but a good reference for the IP networking controls and whatthey do to the kernel.

* The Internet Control Message Protocol - A good, brief document describing the ICMP protocol in detail. Written by Ralph Walden.

* RFC 792 - Internet Control Message Protocol -The definitive resource for all information about ICMP packets. Whatevertechnical information you need about the ICMP protocol, this is where youshould turn first. Written by J. Postel.

* RFC 793 - Transmission Control Protocol - This is the original resource on how TCP should behave on all hosts. This document has been the standard on how TCP should work since 1981 and forward. Extremely technical, but a must read for anyone who wants to learn TCP in every detail. This was originally a Department of Defense standard written by J. Postel.

* ip_dynaddr.txt - from the 2.4.14 kernel. A really shortreference to the ip_dynaddr settings available via sysctl and the procfile system.

* iptables.8 - The iptables 1.2.4 man page. This is an HTMLizedversion of the man page which is an excellent reference when reading/writingiptables rule-sets. Always have it at hand.

* Firewall rules table - A small PDF document gracefully given to thisproject by Stuart Clark, which gives a reference form where you can write all ofthe information needed for your firewall, in a simple manner.

* http://www.netfilter.org/ - The official Netfilter and iptablessite. It is a must for everyone wanting to set up iptablesand Netfilter in linux.

* http://www.netfilter.org/documentation/index.html#FAQ- The official Netfilter Frequently AskedQuestions. Also a good place to start at when wondering whatiptables and Netfilter is about.

* http://www.netfilter.org/unreliable-guides/packet-filtering-HOWTO/index.html- Rusty Russells Unreliable Guide to packet filtering. Excellent documentationabout basic packet filtering with iptables written by one ofthe core developers of iptables andNetfilter.

* http://www.netfilter.org/unreliable-guides/NAT-HOWTO/index.html- Rusty Russells Unreliable Guide to Network Address Translation. Excellentdocumentation about Network Address Translation iniptables and Netfilter written by one ofthe core developers, Rusty Russell.

* http://www.netfilter.org/unreliable-guides/netfilter-hacking-HOWTO/index.html- Rusty Russells Unreliable Netfilter Hacking HOW-TO. One of the fewdocumentations on how to write code in the Netfilter andiptables user-space and kernel space code-base. This was alsowritten by Rusty Russell.

* http://www.linuxguruz.org/iptables/ - Excellent link-page with links to most of the pages on the Internetabout iptables and Netfilter. Alsomaintains a list of iptables scripts fordifferent purposes.

* http://www.islandsoft.net/veerapen.html -Excellent discussion on automatic hardening of iptables andhow to make small changes that will make your computer automatically add hostilesites to a special ban list in iptables.

* /etc/protocols - An example protocols file taken from the Slackware distribution. This can be used to find outwhat protocol number different protocols have, such as the IP, ICMP or TCP protocols have.

* /etc/services - An example services file taken from the Slackware distribution. This is extremely good toget used to reading once in a while, specifically if you want to get a basic lookat what protocols runs on different ports.

* Internet Engineering Task Force - This is one of the biggest groups when it comes to setting and maintaining Internet standards. They are the ones maintaining the RFC repository, and consist of a large group of companies and individuals that work together to ensure the interoperability of the Internet.

* Linux Advanced Routing and Traffic Control HOW-TO - This site hosts the Linux Advanced Routing and Traffic Control HOWTO. It is one of the biggest and best documents regarding Linux advanced routing. Maintained by Bert Hubert.

* Paksecured Linux Kernel patches - A site containing all of the kernel patches written by Matthew G. Marsh. Among others, the FTOS patch is available here.

* ULOGD project page - The homepage of the ULOGD site.

* The Linux Documentation Project is a great site for documentation. Most big documents for Linux is available here, and if not in the TLDP, you will have to search the net very carefully. If there is anything you want to know more about, check this site out.

* http://kalamazoolinux.org/presentations/20010417/conntrack.html - Thispresentation contains an excellent explanation of the conntrack modules andtheir work in Netfilter. If you are interested in more documentation onconntrack, this is a "must read".

* http://www.docum.org -Excellent information about the CBQ,tc and the ip commands in Linux. One ofthe few sites that has any information at all about these programs. Maintainedby Stef Coene.

* http://lists.samba.org/mailman/listinfo/netfilter- The official Netfilter mailing-list. Extremely useful in case you havequestions about something not covered in this document or any of the otherlinks here.

And of course the iptables source, documentation andindividuals who helped me.

AppendixÂ E.Â Acknowledgments

I would like to thank the following people for their help on this document:

* FabriceMarie, For major updates to my horrible grammar and spelling.Also a huge thanks for updating the tutorial to DocBook format with make filesetc.

* MarcBoucher, For helping me out on some aspects on using thestate matching code.

* Frode E.Nyboe, For greatly improving therc.firewall rules and giving great inspiration whilei was to rewrite the rule-set and being the one who introduced the multiple tabletraversing into the same file.

* ChapmanBrad, Alexander W.Janssen, Both for making me realize I was thinking wrongabout how packets traverse the basic NAT and filters tables and in which orderthey show up.

* MichielBrandenburg, MylesUyema, For helping me out with some of the state matchingcode and getting it to work.

* Kent `Artech'Stahre, For helping me out with the graphics. I know I suckat graphics, and you're better than most I know who do graphics;). Also thanksfor checking the tutorial for errors etc.

* Anders 'DeZENT' Johansson, For hinting me aboutstrange ISPs and so on that uses reserved networks on the Internet, or atleast on the Internet for you.

* Jeremy`Spliffy' Smith, For giving me hints at stuff that mightscrew up for people and for trying it out and checking for errors in what I'vewritten.

And of course everyone else I talked to and asked for comments on this file,sorry for not mentioning everyone.

AppendixÂ F.Â History

Version 1.1.20 (25 Nov 2004)
http://iptables-tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Sascha Nitsch (Sash@linuxhowtos.org)
Version 1.1.19 (21 May 2003)
http://iptables-tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Peter van Kampen, Xavier Bartol, Jon Anderson, Thorsten Bremer and Spanish Translation Team.
Version 1.1.18 (24 Apr 2003)
http://iptables-tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Stuart Clark, Robert P. J. Day, Mark Orenstein and Edmond Shwayri.
Version 1.1.17 (6 Apr 2003)
http://iptables-tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Geraldo Amaral Filho, Ondrej Suchy, Dino Conti, Robert P. J. Day, Velev Dimo, Spencer Rouser, Daveonos, Amanda Hickman, Olle Jonsson and Bengt Aspvall.
Version 1.1.16 (16 Dec 2002)
http://iptables-tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Clemens Schwaighower, Uwe Dippel and Dave Wreski.
Version 1.1.15 (13 Nov 2002)
http://iptables-tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Mark Sonarte, A. Lester Buck, Robert P. J. Day, Togan Muftuoglu, Antony Stone, Matthew F. Barnes and Otto Matejka.
Version 1.1.14 (14 Oct 2002)
http://iptables-tutorial.frozentux.net
By: Oskar Andreasson
Contributors: Carol Anne, Manuel Minzoni, Yves Soun, Miernik, Uwe Dippel, Dave Klipec and Eddy L O Jansson.
Version 1.1.13 (22 Aug 2002)
http://iptables-tutorial.haringstad.com
By: Oskar Andreasson
Contributors: Tons of people reporting bad HTML version.
Version 1.1.12 (19 Aug 2002)
http://www.netfilter.org/tutorial/
By: Oskar Andreasson
Contributors: Peter Schubnell, Stephen J. Lawrence, Uwe Dippel, Bradley Dilger, Vegard Engen, Clifford Kite, Alessandro Oliveira, Tony Earnshaw,

Harald Welte, Nick Andrew and Stepan Kasal.

Version 1.1.11 (27 May 2002)

http://www.netfilter.org/tutorial/

By: Oskar Andreasson

Contributors: Steve Hnizdur, Lonni Friedman, Jelle Kalf, Harald Welte, Valentina Barrios and Tony Earnshaw.

Version 1.1.10 (12 April 2002)

http://www.boingworld.com/workshops/linux/iptables-tutorial/

By: Oskar Andreasson

Contributors: Jelle Kalf, Theodore Alexandrov, Paul Corbett, Rodrigo Rubira Branco, Alistair Tonner, Matthew G. Marsh, Uwe Dippel, Evan Nemerson and Marcel J.E. Mol.

Version 1.1.9 (21 March 2002)

http://www.boingworld.com/workshops/linux/iptables-tutorial/

By: Oskar Andreasson

Contributors: Vince Herried, Togan Muftuoglu, Galen Johnson, Kelly Ashe, Janne Johansson, Thomas Smets, Peter Horst, Mitch Landers, Neil Jolly, Jelle Kalf, Jason Lam and Evan Nemerson.

Version 1.1.8 (5 March 2002)

http://www.boingworld.com/workshops/linux/iptables-tutorial/

By: Oskar Andreasson

Version 1.1.7 (4 February 2002)

http://www.boingworld.com/workshops/linux/iptables-tutorial/

By: Oskar Andreasson

Contributors: Parimi Ravi, Phil Schultz, Steven McClintoc, Bill Dossett, Dave Wreski, Erik Sjölund, Adam Mansbridge, Vasoo Veerapen, Aladdin and Rusty Russell.

Version 1.1.6 (7 December 2001)

http://people.unix-fu.org/andreasson/

By: Oskar Andreasson

Contributors: Jim Ramsey, Phil Schultz, Göran Båge, Doug Monroe, Jasper Aikema, Kurt Lieber, Chris Tallon, Chris Martin, Jonas Pasche, Jan Labanowski, Rodrigo R. Branco, Jacco van Koll and Dave Wreski.

Version 1.1.5 (14 November 2001)

http://people.unix-fu.org/andreasson/

By: Oskar Andreasson

Contributors: Fabrice Marie, Merijn Schering and Kurt Lieber.

Version 1.1.4 (6 November 2001)

http://people.unix-fu.org/andreasson

By: Oskar Andreasson

Contributors: Stig W. Jensen, Steve Hnizdur, Chris Pluta and Kurt Lieber.

Version 1.1.3 (9 October 2001)

http://people.unix-fu.org/andreasson

By: Oskar Andreasson

Contributors:Â JoniÂ Chu,Â N.EmileÂ Akabi-DavisÂ andÂ JelleÂ Kalf.

VersionÂ 1.1.2Â (29Â SeptemberÂ 2001)

http://people.unix-fu.org/andreasson

By:Â OskarÂ Andreasson

VersionÂ 1.1.1Â (26Â SeptemberÂ 2001)

http://people.unix-fu.org/andreasson

By:Â OskarÂ Andreasson

Contributors:Â DaveÂ Richardson.

VersionÂ 1.1.0Â (15Â SeptemberÂ 2001)

http://people.unix-fu.org/andreasson

By:Â OskarÂ Andreasson

VersionÂ 1.0.9Â (9Â SeptemberÂ 2001)

http://people.unix-fu.org/andreasson

By:Â OskarÂ Andreasson

VersionÂ 1.0.8Â (7Â SeptemberÂ 2001)

http://people.unix-fu.org/andreasson

By:Â OskarÂ Andreasson

VersionÂ 1.0.7Â (23Â AugustÂ 2001)

http://people.unix-fu.org/andreasson

By:Â OskarÂ Andreasson

Contributors:Â FabriceÂ Marie.

VersionÂ 1.0.6

http://people.unix-fu.org/andreasson

By:Â OskarÂ Andreasson

VersionÂ 1.0.5

http://people.unix-fu.org/andreasson

By:Â OskarÂ Andreasson

Contributors:Â FabriceÂ Marie.

Â Â Â

AppendixÂ G.Â GNU Free Documentation License

Table of Contents
0. PREAMBLE1. APPLICABILITY AND DEFINITIONS2. VERBATIM COPYING3. COPYING IN QUANTITY4. MODIFICATIONS5. COMBINING DOCUMENTS6. COLLECTIONS OF DOCUMENTS7. AGGREGATION WITH INDEPENDENT WORKS8. TRANSLATION9. TERMINATION10. FUTURE REVISIONS OF THIS LICENSEHow to use this License for your documents

Version 1.1, March 2000

Copyright (C) 2000 Free Software Foundation, Inc.59 Temple Place, Suite 330, Boston, MA 02111-1307 USAEveryone is permitted to copy and distribute verbatim copiesof this license document, but changing it is not allowed.

PREAMBLE

The purpose of this License is to make a manual, textbook,   or other written document "free" in the sense of freedom: to   assure everyone the effective freedom to copy and redistribute it,   with or without modifying it, either commercially or   noncommercially. Secondarily, this License preserves for the   author and publisher a way to get credit for their work, while not   being considered responsible for modifications made by   others.

This License is a kind of "copyleft", which means that   derivative works of the document must themselves be free in the   same sense. It complements the GNU General Public License, which   is a copyleft license designed for free software.

We have designed this License in order to use it for manuals   for free software, because free software needs free documentation: a free program should come with manuals providing the same   freedoms that the software does. But this License is not limited   to software manuals; it can be used for any textual work,   regardless of subject matter or whether it is published as a   printed book. We recommend this License principally for works   whose purpose is instruction or reference.

APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that   contains a notice placed by the copyright holder saying it can be   distributed under the terms of this License. The "Document",   below, refers to any such manual or work. Any member of the   public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work   containing the Document or a portion of it, either copied   verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter   section of the Document that deals exclusively with the   relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains   nothing that could fall directly within that overall subject.   (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.)   The relationship could be a matter of historical connection with   the subject or with related matters, or of legal, commercial,   philosophical, ethical or political position regarding   them.

The "Invariant Sections" are certain Secondary Sections   whose titles are designated, as being those of Invariant Sections,   in the notice that says that the Document is released under this   License.

The "Cover Texts" are certain short passages of text that   are listed, as Front-Cover Texts or Back-Cover Texts, in the   notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a   machine-readable copy, represented in a format whose specification   is available to the general public, whose contents can be viewed   and edited directly and straightforwardly with generic text   editors or (for images composed of pixels) generic paint programs   or (for drawings) some widely available drawing editor, and that   is suitable for input to text formatters or for automatic   translation to a variety of formats suitable for input to text   formatters. A copy made in an otherwise Transparent file format   whose markup has been designed to thwart or discourage subsequent   modification by readers is not Transparent. A copy that is not   is called "Opaque".

Examples of suitable formats for Transparent copies include   plain ASCII without markup, Texinfo input format, LaTeX input   format, SGML or XML using a publicly available DTD, and   standard-conforming simple HTML designed for human modification.   Opaque formats include PostScript, PDF, proprietary formats that   can be read and edited only by proprietary word processors, SGML   or XML for which the DTD and/or processing tools are not generally   available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page   itself, plus such following pages as are needed to hold, legibly,   the material this License requires to appear in the title page.   For works in formats which do not have any title page as such,   "Title Page" means the text near the most prominent appearance of   the work's title, preceding the beginning of the body of the   text.

VERBATIM COPYING

You may copy and distribute the Document in any medium,   either commercially or noncommercially, provided that this   License, the copyright notices, and the license notice saying this   License applies to the Document are reproduced in all copies, and   that you add no other conditions whatsoever to those of this   License. You may not use technical measures to obstruct or   control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for   copies. If you distribute a large enough number of copies you   must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated   above, and you may publicly display copies.

COPYING IN QUANTITY

If you publish printed copies of the Document numbering more   than 100, and the Document's license notice requires Cover Texts,   you must enclose the copies in covers that carry, clearly and   legibly, all these Cover Texts: Front-Cover Texts on the front   cover, and Back-Cover Texts on the back cover. Both covers must   also clearly and legibly identify you as the publisher of these   copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add   other material on the covers in addition. Copying with changes   limited to the covers, as long as they preserve the title of the   Document and satisfy these conditions, can be treated as verbatim   copying in other respects.

If the required texts for either cover are too voluminous to   fit legibly, you should put the first ones listed (as many as fit   reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document   numbering more than 100, you must either include a   machine-readable Transparent copy along with each Opaque copy,   or   state in or with each Opaque copy a publicly-accessible   computer-network location containing a complete Transparent copy   of the Document, free of added material, which the general   network-using public has access to download anonymously at no   charge using public-standard network protocols. If you use the   latter option, you must take reasonably prudent steps, when you   begin distribution of Opaque copies in quantity, to ensure that   this Transparent copy will remain thus accessible at the stated   location until at least one year after the last time you   distribute an Opaque copy (directly or through your agents or   retailers) of that edition to the public.

It is requested, but not required, that you contact the   authors of the Document well before redistributing any large   number of copies, to give them a chance to provide you with an updated version of the Document.

MODIFICATIONS

You may copy and distribute a Modified Version of the   Document under the conditions of sections 2 and 3 above, provided   that you release the Modified Version under precisely this   License, with the Modified Version filling the role of the   Document, thus licensing distribution and modification of the   Modified Version to whoever possesses a copy of it. In addition,   you must do these things in the Modified Version:

* Use in the Title Page   (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if   there were any, be listed in the History section of the   Document). You may use the same title as a previous version if   the original publisher of that version gives permission.

* List on the Title Page,   as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version,   together with at least five of the principal authors of the   Document (all of its principal authors, if it has less than   five).

* State on the Title page   the name of the publisher of the Modified Version, as the publisher.

* Preserve all the   copyright notices of the Document.

* Add an appropriate   copyright notice for your modifications adjacent to the other copyright notices.

* Include, immediately   after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this   License, in the form shown in the Addendum below.

* Preserve in that license   notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

* Include an unaltered   copy of this License.

* Preserve the section   entitled "History", and its title, and add to it an item stating   at least the title, year, new authors, and publisher of the   Modified Version as given on the Title Page. If there is no   section entitled "History" in the Document, create one stating   the title, year, authors, and publisher of the Document as given   on its Title Page, then add an item describing the Modified   Version as stated in the previous sentence.

* Preserve the network   location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network   locations given in the Document for previous versions it was   based on. These may be placed in the "History" section. You   may omit a network location for a work that was published at   least four years before the Document itself, or if the original   publisher of the version it refers to gives permission.

* In any section entitled   "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of   each of the contributor acknowledgements and/or dedications   given therein.

* Preserve all the   Invariant Sections of the Document, unaltered in their text and   in their titles. Section numbers or the equivalent are not   considered part of the section titles.

* Delete any section   entitled "Endorsements". Such a section may not be included in the Modified Version.

* Do not retitle any   existing section as "Endorsements" or to conflict in title with   any Invariant Section.

If the Modified Version includes new front-matter sections   or appendices that qualify as Secondary Sections and contain no   material copied from the Document, you may at your option   designate some or all of these sections as invariant. To do this,   add their titles to the list of Invariant Sections in the Modified   Version's license notice. These titles must be distinct from any   other section titles.

You may add a section entitled "Endorsements", provided it   contains nothing but endorsements of your Modified Version by   various parties--for example, statements of peer review or that   the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover   Text, and a passage of up to 25 words as a Back-Cover Text, to the   end of the list of Cover Texts in the Modified Version. Only one   passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the   Document already includes a cover text for the same cover,   previously added by you or by arrangement made by the same entity   you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous   publisher that added the old one.

The author(s) and publisher(s) of the Document do not by   this License give permission to use their names for publicity for   or to assert or imply endorsement of any Modified Version.

COMBINING DOCUMENTS

You may combine the Document with other documents released   under this License, under the terms defined in section 4 above for   modified versions, provided that you include in the combination   all of the Invariant Sections of all of the original documents,   unmodified, and list them all as Invariant Sections of your   combined work in its license notice.

The combined work need only contain one copy of this   License, and multiple identical Invariant Sections may be replaced   with a single copy. If there are multiple Invariant Sections with   the same name but different contents, make the title of each such   section unique by adding at the end of it, in parentheses, the   name of the original author or publisher of that section if known,   or else a unique number. Make the same adjustment to the section   titles in the list of Invariant Sections in the license notice of   the combined work.

In the combination, you must combine any sections entitled   "History" in the various original documents, forming one section   entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You   must delete all sections entitled "Endorsements."

COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and   other documents released under this License, and replace the   individual copies of this License in the various documents with a   single copy that is included in the collection, provided that you   follow the rules of this License for verbatim copying of each of   the documents in all other respects.

You may extract a single document from such a collection,   and distribute it individually under this License, provided you   insert a copy of this License into the extracted document, and   follow this License in all other respects regarding verbatim   copying of that document.

AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other   separate and independent documents or works, in or on a volume of   a storage or distribution medium, does not as a whole count as a   Modified Version of the Document, provided no compilation   copyright is claimed for the compilation. Such a compilation is   called an "aggregate", and this License does not apply to the   other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves   derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to   these copies of the Document, then if the Document is less than   one quarter of the entire aggregate, the Document's Cover Texts   may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole   aggregate.

TRANSLATION

Translation is considered a kind of modification, so you may   distribute translations of the Document under the terms of section   4. Replacing Invariant Sections with translations requires   special permission from their copyright holders, but you may   include translations of some or all Invariant Sections in addition   to the original versions of these Invariant Sections. You may   include a translation of this License provided that you also   include the original English version of this License. In case of   a disagreement between the translation and the original English   version of this License, the original English version will prevail.

TERMINATION

You may not copy, modify, sublicense, or distribute the   Document except as expressly provided for under this License. Any   other attempt to copy, modify, sublicense or distribute the   Document is void, and will automatically terminate your rights   under this License. However, parties who have received copies, or   rights, from you under this License will not have their licenses   terminated so long as such parties remain in full   compliance.

FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised   versions of the GNU Free Documentation License from time to time.   Such new versions will be similar in spirit to the present   version, but may differ in detail to address new problems or   concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing   version number. If the Document specifies that a particular   numbered version of this License "or any later version" applies to   it, you have the option of following the terms and conditions   either of that specified version or of any later version that has   been published (not as a draft) by the Free Software Foundation.   If the Document does not specify a version number of this License,   you may choose any version ever published (not as a draft) by the   Free Software Foundation.

How to use this License for your documents

To use this License in a document you have written, include   a copy of the License in the document and put the following   copyright and license notices just after the title page:

      Copyright (c) YEAR YOUR NAME.   Permission is granted to copy, distribute and/or modify this document   under the terms of the GNU Free Documentation License, Version 1.1   or any later version published by the Free Software Foundation;   with the Invariant Sections being LIST THEIR TITLES, with the   Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.   A copy of the license is included in the section entitled "GNU   Free Documentation License".

If you have no Invariant Sections, write "with no Invariant   Sections" instead of saying which ones are invariant. If you have   no Front-Cover Texts, write "no Front-Cover Texts" instead of   "Front-Cover Texts being LIST"; likewise for Back-Cover   Texts.

If your document contains nontrivial examples of program   code, we recommend releasing these examples in parallel under your   choice of free software license, such as the GNU General Public   License, to permit their use in free software.

AppendixÂ H.Â GNU General Public License

Table of Contents
0. Preamble1. TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION2. How to Apply These Terms to Your New Programs

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.59 Temple Place, Suite 330, Boston, MA 02111-1307 USAEveryone is permitted to copy and distribute verbatim copiesof this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away yourfreedom to share and change it. By contrast, the GNU General PublicLicense is intended to guarantee your freedom to share and change freesoftware--to make sure the software is free for all its users. ThisGeneral Public License applies to most of the Free SoftwareFoundation's software and to any other program whose authors commit tousing it. (Some other Free Software Foundation software is covered bythe GNU Library General Public License instead.) You can apply it toyour programs, too.

When we speak of free software, we are referring to freedom, notprice. Our General Public Licenses are designed to make sure that youhave the freedom to distribute copies of free software (and charge forthis service if you wish), that you receive source code or can get itif you want it, that you can change the software or use pieces of itin new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions thatforbid anyone to deny you these rights or to ask you to surrender therights. These restrictions translate to certain responsibilities for youif you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program,whether gratis or for a fee, you must give the recipients all the rightsthat you have. You must make sure that they, too, receive or can get thesource code. And you must show them these terms so they know theirrights.

We protect your rights with two steps: (1) copyright thesoftware, and (2) offer you this license which gives you legal permissionto copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to makecertain that everyone understands that there is no warranty for this freesoftware. If the software is modified by someone else and passed on, wewant its recipients to know that what they have is not the original, sothat any problems introduced by others will not reflect on the originalauthors' reputations.

Finally, any free program is threatened constantly bysoftware patents. We wish to avoid the danger that redistributors of afree program will individually obtain patent licenses, in effect makingthe program proprietary. To prevent this, we have made it clear that anypatent must be licensed for everyone's free use or not licensed atall.

The precise terms and conditions for copying, distributionand modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

* This License applies to any program or other workwhich contains a notice placed by the copyright holder saying it may bedistributed under the terms of this General Public License. The"Program", below, refers to any such program or work, and a "work based onthe Program" means either the Program or any derivative work undercopyright law: that is to say, a work containing the Program or a portionof it, either verbatim or with modifications and/or translated intoanother language. (Hereinafter, translation is included withoutlimitation in the term "modification".) Each licensee is addressed as"you".

Activities other than copying, distribution and modificationare not covered by this License; they are outside its scope. The act ofrunning the Program is not restricted, and the output from the Program iscovered only if its contents constitute a work based on the Program(independent of having been made by running the Program). Whether that istrue depends on what the Program does.

* You may copy and distribute verbatim copies of theProgram's source code as you receive it, in any medium, provided that youconspicuously and appropriately publish on each copy an appropriatecopyright notice and disclaimer of warranty; keep intact all the noticesthat refer to this License and to the absence of any warranty; and giveany other recipients of the Program a copy of this License along with theProgram.

    You may charge a fee for the physical act of transferring acopy, and you may at your option offer warranty protection in exchange fora fee.

* You may modify your copy or copies of the Programor any portion of it, thus forming a work based on the Program, and copyand distribute such modifications or work under the terms of Section 1above, provided that you also meet all of these conditions:

* You must cause the modified files to carryprominent notices stating that you changed the files and the date of anychange.

* You must cause any work that you distribute or publish,that in whole or in part contains or is derived from the Program or anypart thereof, to be licensed as a whole at no charge to all third partiesunder the terms of this License.

* If the modified program normally reads commandsinteractively when run, you must cause it, when started running for suchinteractive use in the most ordinary way, to print or display anannouncement including an appropriate copyright notice and a notice thatthere is no warranty (or else, saying that you provide a warranty) andthat users may redistribute the program under these conditions, andtelling the user how to view a copy of this License. (Exception: if theProgram itself is interactive but does not normally print such anannouncement, your work based on the Program is not required to print anannouncement.)

    These requirements apply to the modified work as a whole. Ifidentifiable sections of that work are not derived from the Program, andcan be reasonably considered independent and separate works in themselves,then this License, and its terms, do not apply to those sections when youdistribute them as separate works. But when you distribute the samesections as part of a whole which is a work based on the Program, thedistribution of the whole must be on the terms of this License, whosepermissions for other licensees extend to the entire whole, and thus toeach and every part regardless of who wrote it.

    Thus, it is not the intent of this section to claim rights orcontest your rights to work written entirely by you; rather, the intent isto exercise the right to control the distribution of derivative orcollective works based on the Program.

    In addition, mere aggregation of another work not based on theProgram with the Program (or with a work based on the Program) on a volumeof a storage or distribution medium does not bring the other work underthe scope of this License.

* You may copy and distribute the Program (or a workbased on it, under Section 2) in object code or executable form under theterms of Sections 1 and 2 above provided that you also do one of thefollowing:

* Accompany it with the completecorresponding machine-readable source code, which must be distributedunder the terms of Sections 1 and 2 above on a medium customarily used forsoftware interchange; or,

* Accompany it with a written offer, validfor at least three years, to give any third party, for a charge no morethan your cost of physically performing source distribution, a completemachine-readable copy of the corresponding source code, to be distributedunder the terms of Sections 1 and 2 above on a medium customarily used forsoftware interchange; or,

* Accompany it with the information youreceived as to the offer to distribute corresponding source code. (Thisalternative is allowed only for noncommercial distribution and only if youreceived the program in object code or executable form with such an offer,in accord with Subsection b above.)

The source code for a work means the preferred form of thework for making modifications to it. For an executable work, completesource code means all the source code for all modules it contains, plusany associated interface definition files, plus the scripts used tocontrol compilation and installation of the executable. However, as aspecial exception, the source code distributed need not include anythingthat is normally distributed (in either source or binary form) with themajor components (compiler, kernel, and so on) of the operating system onwhich the executable runs, unless that component itself accompanies theexecutable.

If distribution of executable or object code is made byoffering access to copy from a designated place, then offering equivalentaccess to copy the source code from the same place counts as distributionof the source code, even though third parties are not compelled to copythe source along with the object code.

* You may not copy, modify, sublicense, ordistribute the Program except as expressly provided under this License.Any attempt otherwise to copy, modify, sublicense or distribute theProgram is void, and will automatically terminate your rights under thisLicense. However, parties who have received copies, or rights, from youunder this License will not have their licenses terminated so long as suchparties remain in full compliance.

* You are not required to accept this License, sinceyou have not signed it. However, nothing else grants you permission tomodify or distribute the Program or its derivative works. These actionsare prohibited by law if you do not accept this License. Therefore, bymodifying or distributing the Program (or any work based on the Program),you indicate your acceptance of this License to do so, and all its termsand conditions for copying, distributing or modifying the Program or worksbased on it.

* Each time you redistribute the Program (or anywork based on the Program), the recipient automatically receives a licensefrom the original licensor to copy, distribute or modify the Programsubject to these terms and conditions. You may not impose any furtherrestrictions on the recipients' exercise of the rights granted herein. Youare not responsible for enforcing compliance by third parties to thisLicense.

* If, as a consequence of a court judgment orallegation of patent infringement or for any other reason (not limited topatent issues), conditions are imposed on you (whether by court order,agreement or otherwise) that contradict the conditions of this License,they do not excuse you from the conditions of this License. If you cannotdistribute so as to satisfy simultaneously your obligations under thisLicense and any other pertinent obligations, then as a consequence you maynot distribute the Program at all. For example, if a patent license wouldnot permit royalty-free redistribution of the Program by all those whoreceive copies directly or indirectly through you, then the only way youcould satisfy both it and this License would be to refrain entirely fromdistribution of the Program.

If any portion of this section is held invalid orunenforceable under any particular circumstance, the balance of thesection is intended to apply and the section as a whole is intended toapply in other circumstances.

It is not the purpose of this section to induce you toinfringe any patents or other property right claims or to contest validityof any such claims; this section has the sole purpose of protecting theintegrity of the free software distribution system, which is implementedby public license practices. Many people have made generous contributionsto the wide range of software distributed through that system in relianceon consistent application of that system; it is up to the author/donor todecide if he or she is willing to distribute software through any othersystem and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what isbelieved to be a consequence of the rest of this License.

If the distribution and/or use of the Program is restrictedin certain countries either by patents or by copyrighted interfaces, theoriginal copyright holder who places the Program under this License mayadd an explicit geographical distribution limitation excluding thosecountries, so that distribution is permitted only in or among countriesnot thus excluded. In such case, this License incorporates the limitationas if written in the body of this License.

* The Free Software Foundation may publish revisedand/or new versions of the General Public License from time to time.Such new versions will be similar in spirit to the present version, butmay differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. Ifthe Program specifies a version number of this License which applies to itand "any later version", you have the option of following the terms andconditions either of that version or of any later version published by theFree Software Foundation. If the Program does not specify a versionnumber of this License, you may choose any version ever published by theFree Software Foundation.

* If you wish to incorporate parts of the Programinto other free programs whose distribution conditions are different,write to the author to ask for permission. For software which iscopyrighted by the Free Software Foundation, write to the Free SoftwareFoundation; we sometimes make exceptions for this. Our decision will beguided by the two goals of preserving the free status of all derivativesof our free software and of promoting the sharing and reuse of softwaregenerally.

* NO WARRANTY

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE,THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BYAPPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHTHOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTYOF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULARPURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAMIS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OFALL NECESSARY SERVICING, REPAIR OR CORRECTION.

* IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW ORAGREED TO IN WRITINGWILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/ORREDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISINGOUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITEDTO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BYYOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHERPROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THEPOSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of thegreatest possible use to the public, the best way to achieve this is tomake it free software which everyone can redistribute and change underthese terms.

To do so, attach the following notices to the program. Itis safest to attach them to the start of each source file to mosteffectively convey the exclusion of warranty; and each file should have atleast the "copyright" line and a pointer to where the full notice isfound.

>oneÂ lineÂ toÂ giveÂ theÂ program'sÂ nameÂ andÂ aÂ briefÂ ideaÂ ofÂ whatÂ itÂ does.> CopyrightÂ (C)Â >year>Â Â >nameÂ ofÂ author>
Â Â Â Â

This program is free software; you can redistribute it and/or modify   it under the terms of the GNU General Public License as published by   the Free Software Foundation; either version 2 of the License, or   (at your option) any later version.

This program is distributed in the hope that it will be useful,   but WITHOUT ANY WARRANTY; without even the implied warranty of   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the   GNU General Public License for more details.

You should have received a copy of the GNU General Public License   along with this program; if not, write to the Free Software   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and papermail.

If the program is interactive, make it output a short notice likethis when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author   Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.    This is free software, and you are welcome to redistribute it      under certain conditions; type `show c' for details.

 The hypothetical commands `show w' and `show c' should showthe appropriate parts of the General Public License. Of course, thecommands you use may be called something other than `show w' and `show c';they could even be mouse-clicks or menu items--whatever suits yourprogram.

 You should also get your employer (if you work as aprogrammer) or your school, if any, to sign a "copyright disclaimer" forthe program, if necessary. Here is a sample; alter the names:

Yoyodyne,Â Inc.,Â herebyÂ disclaimsÂ allÂ copyrightÂ interestÂ inÂ theÂ program `Gnomovision'Â (whichÂ makesÂ passesÂ atÂ compilers)Â writtenÂ byÂ JamesÂ Hacker.
Â Â

>signatureÂ ofÂ TyÂ Coon>,Â 1Â AprilÂ 1989
TyÂ Coon,Â PresidentÂ ofÂ Vice
Â Â

This General Public License does not permit incorporating your program intoproprietary programs. If your program is a subroutine library, you mayconsider it more useful to permit linking proprietary applications with thelibrary. If this is what you want to do, use the GNU Library GeneralPublic License instead of this License.

AppendixÂ I.Â Example scripts code-base

Table of Contents

Example rc.firewall script

```sh
#!/bin/sh
#
# rc.firewall - Initial SIMPLE IP Firewall script for Linux 2.4.x and iptables
#
# Copyright (C) 2001  Oskar Andreasson >bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59 Temple
# Place, Suite 330, Boston, MA  02111-1307   USA
#

###########################################################################
#
# 1. Configuration options.
#

#
# 1.1 Internet Configuration.
#

INET_IP="194.236.50.155"
INET_IFACE="eth0"
INET_BROADCAST="194.236.50.255"


#
# 1.1.1 DHCP
#


#
# 1.1.2 PPPoE
#
```

```sh
#
# 1.2 Local Area Network configuration.
#
# your LAN's IP range and localhost IP. /24 means to only use the first 24
```

Example rc.DMZ.firewall script

```
#!/bin/sh
#
# rc.DMZ.firewall - DMZ IP Firewall script for Linux 2.4.x and iptables
#
# Copyright (C) 2001  Oskar Andreasson >bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59 Temple
# Place, Suite 330, Boston, MA  02111-1307   USA
#

#########################################################################
#
# 1. Configuration options.
#


#
# 1.1 Internet Configuration.
#

INET_IP="194.236.50.152"
HTTP_IP="194.236.50.153"
DNS_IP="194.236.50.154"
INET_IFACE="eth0"


#
# 1.1.1 DHCP
#


#
# 1.1.2 PPPoE
#


#
# 1.2 Local Area Network configuration.
#
# your LAN's IP range and localhost IP. /24 means to only use the first 24
# bits of the 32 bit IP address. the same as netmask 255.255.255.0
#

LAN_IP="192.168.0.1"
LAN_IFACE="eth1"


#
# 1.3 DMZ Configuration.
#

DMZ_HTTP_IP="192.168.1.2"
DMZ_DNS_IP="192.168.1.3"
DMZ_IP="192.168.1.1"
```

Example rc.UTIN.firewall script

```
#!/bin/sh
#
# rc.firewall - UTIN Firewall script for Linux 2.4.x and iptables
#
# Copyright (C) 2001  Oskar Andreasson >bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59 Temple
# Place, Suite 330, Boston, MA  02111-1307   USA
#

##########################################################################
#
# 1. Configuration options.
#


#
# 1.1 Internet Configuration.
#

INET_IP="194.236.50.155"
INET_IFACE="eth0"
INET_BROADCAST="194.236.50.255"


#
# 1.1.1 DHCP
#


#
# 1.1.2 PPPoE
#


#
# 1.2 Local Area Network configuration.
#
# your LAN's IP range and localhost IP. /24 means to only use the first 24
# bits of the 32 bit IP address. the same as netmask 255.255.255.0
#

LAN_IP="192.168.0.2"
LAN_IP_RANGE="192.168.0.0/16"
LAN_IFACE="eth1"


#
# 1.3 DMZ Configuration.
#


#
# 1.4 Localhost Configuration.
#
```

Example rc.DHCP.firewall script

```
#!/bin/sh
#
# rc.firewall - DHCP IP Firewall script for Linux 2.4.x and iptables
#
# Copyright (C) 2001  Oskar Andreasson >bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59 Temple
# Place, Suite 330, Boston, MA  02111-1307   USA
#

########################################################################
#
# 1. Configuration options.
#


#
# 1.1 Internet Configuration.
#

INET_IFACE="eth0"


#
# 1.1.1 DHCP
#


#
# Information pertaining to DHCP over the Internet, if needed.
#
# Set DHCP variable to no if you don't get IP from DHCP. If you get DHCP
# over the Internet set this variable to yes, and set up the proper IP
# address for the DHCP server in the DHCP_SERVER variable.
#

DHCP="no"
DHCP_SERVER="195.22.90.65"


#
# 1.1.2 PPPoE
#


# Configuration options pertaining to PPPoE.
#
# If you have problem with your PPPoE connection, such as large mails not
# getting through while small mail get through properly etc, you may set
# this option to "yes" which may fix the problem. This option will set a
# rule in the PREROUTING chain of the mangle table which will clamp
# (resize) all routed packets to PMTU (Path Maximum Transmit Unit).
#
# Note that it is better to set this up in the PPPoE package itself, since
```

Example rc.flush-iptables script

```
#!/bin/sh
#
# rc.flush-iptables - Resets iptables to default values.
#
# Copyright (C) 2001  Oskar Andreasson >bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59 Temple
# Place, Suite 330, Boston, MA  02111-1307   USA


#
# Configurations
#
IPTABLES="/usr/sbin/iptables"


#
# reset the default policies in the filter table.
#
$IPTABLES -P INPUT ACCEPT
$IPTABLES -P FORWARD ACCEPT
$IPTABLES -P OUTPUT ACCEPT


#
# reset the default policies in the nat table.
#
$IPTABLES -t nat -P PREROUTING ACCEPT
$IPTABLES -t nat -P POSTROUTING ACCEPT
$IPTABLES -t nat -P OUTPUT ACCEPT


#
# reset the default policies in the mangle table.
#
$IPTABLES -t mangle -P PREROUTING ACCEPT
$IPTABLES -t mangle -P OUTPUT ACCEPT


#
# flush all the rules in the filter and nat tables.
#
$IPTABLES -F
$IPTABLES -t nat -F
$IPTABLES -t mangle -F
#
# erase all chains that's not default in filter and nat table.
#
$IPTABLES -X
$IPTABLES -t nat -X
$IPTABLES -t mangle -X
```

Example rc.test-iptables script

```
#!/bin/bash
#
# rc.test-iptables - test script for iptables chains and tables.
#
# Copyright (C) 2001  Oskar Andreasson >bluefluxATkoffeinDOTnet>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59 Temple
# Place, Suite 330, Boston, MA  02111-1307   USA
#


#
# Filter table, all chains
#
iptables -t filter -A INPUT -p icmp --icmp-type echo-request \
-j LOG --log-prefix="filter INPUT:"
iptables -t filter -A INPUT -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="filter INPUT:"
iptables -t filter -A OUTPUT -p icmp --icmp-type echo-request \
-j LOG --log-prefix="filter OUTPUT:"
iptables -t filter -A OUTPUT -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="filter OUTPUT:"
iptables -t filter -A FORWARD -p icmp --icmp-type echo-request \
-j LOG --log-prefix="filter FORWARD:"
iptables -t filter -A FORWARD -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="filter FORWARD:"


#
# NAT table, all chains except OUTPUT which don't work.
#
iptables -t nat -A PREROUTING -p icmp --icmp-type echo-request \
-j LOG --log-prefix="nat PREROUTING:"
iptables -t nat -A PREROUTING -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="nat PREROUTING:"
iptables -t nat -A POSTROUTING -p icmp --icmp-type echo-request \
-j LOG --log-prefix="nat POSTROUTING:"
iptables -t nat -A POSTROUTING -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="nat POSTROUTING:"
iptables -t nat -A OUTPUT -p icmp --icmp-type echo-request \
-j LOG --log-prefix="nat OUTPUT:"
iptables -t nat -A OUTPUT -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="nat OUTPUT:"


#
# Mangle table, all chains
#
iptables -t mangle -A PREROUTING -p icmp --icmp-type echo-request \
-j LOG --log-prefix="mangle PREROUTING:"
iptables -t mangle -A PREROUTING -p icmp --icmp-type echo-reply \
-j LOG --log-prefix="mangle PREROUTING:"
```

rate this article:
current rating:
Your rating:

image:rdf newsfeed / //static.linuxhowtos.org/data/rdf.png (null)
 |
image:rss newsfeed / //static.linuxhowtos.org/data/rss.png (null)
 |
image:Atom newsfeed / //static.linuxhowtos.org/data/atom.png (null)
- Powered by
image:LeopardCMS / //static.linuxhowtos.org/data/leopardcms.png (null)
 - Running on
image:Gentoo / //static.linuxhowtos.org/data/gentoo.png (null)
 -
Copyright 2004-2020 Sascha Nitsch Unternehmensberatung GmbH
image:Valid XHTML1.1 / //static.linuxhowtos.org/data/xhtml.png (null)
 :
image:Valid CSS / //static.linuxhowtos.org/data/css.png (null)
 :
image:buttonmaker / //static.linuxhowtos.org/data/buttonmaker.png (null)
- Level Triple-A Conformance to Web Content Accessibility Guidelines 1.0 -
- Copyright and legal notices -
Time to create this page: ms
<!--
image:system status display / /status/output.jpg (null)
-->