Linux IP Masquerade HOWTO

David A. Ranch

dranch@trinnet.net

November 13, 2005

This document describes how to enable the Linux IP Masquerade feature on a given Linux host. IP Masquerade is a form of Network Address Translation or NAT which NAT allows internally connected computers that do not have one or more registered Internet IP addresses to communicate to the Internet via the Linux server's Internet IP address.

Table of Contents

Chapter 1. Introduction

1.1. Introduction to IP Masquerading or IP MASQ

This document describes how to enable the Linux IP Masquerade feature on a given Linux host. IP Masquerade, called "IPMASQ" or "MASQ" for short, is a form of Network Address Translation (NAT) which allows internally connected computers that do not have one or more registered Internet IP addresses to communicate to the Internet via the Linux server's Internet IP address. Since IPMASQ is a generic technology, you can connect the Linux server's internal and external to other computers through LAN technologies like Ethernet, TokenRing, and FDDI, as well as dialup connections line PPP or SLIP links. This document primarily uses Ethernet and PPP connections in examples because it is most commonly used with DSL / Cablemodems and dialup connections.

"**This document is intended for systems running stable Linux kernels like 2.4.x, 2.2.x, and 2.0.x preferably on an IBM-compatible PC. IP Masquerade does work on other Linux-supported platforms like Sparc, Alpha, PowerPC, etc. but this HOWTO doesn't cover them in as much detail. Beta kernels such as 2.5.x, 2.3.x, 2.1.x, and ANY kernels less than 2.0.x are NOT covered in this document. The primary reason for this is because many of the older kernels are considered broken. If you are using an older kernel version, it is highly advisable to upgrade to one of the stable Linux kernels before using IP Masquerading.** "

1.2. Foreword, Feedback & Credits

From the original IPMASQ HOWTO author:

"As a new user, I found it very confusing to setup IP masquerade on the Linux kernel, (back then, its was a 1.2.x kernel). Although there was a FAQ and a mailing list, there was no documentation dedicated to this. There was also some requests on the mailing list for a HOWTO manual. So, I decided to write this HOWTO as a starting point for new users and possibly create a building block for other knowledgeable users. If you, the reader, have any additional ideas, corrections, or questions about this document, please feel free to contact us. "

This document was originally written by Ambrose Au back in August, 1996, based on the 1.x kernel IPMASQ FAQ written by Ken Eves and numerous helpful messages from the original IP Masquerade mailing list. In particular, a mailing list message from Matthew Driver inspired Ambrose to set up IP Masquerade and eventually write version 0.80 of this HOWTO. In April 1997,

Ambrose created the Linux IP Masquerade Resource Web site at http://ipmasq.webhop.net which has provided up-to-date information on Linux IP Masquerading ever since. In February 1999, David Ranch took over maintenance of the HOWTO. David then re-wrote the HOWTO and added a substantial number of sections to the document. Today, the HOWTO is still maintained by David where he constantly updates it and fixes any reported bugs, etc.

Please feel free to send any feedback or comments regarding this HOWTO to dranch@trinnet.net if you have any corrections or if any information/URLs/etc. is missing. Your invaluable feedback will certainly influence the future of this HOWTO!

This HOWTO is meant to be a fairly comprehensive guide to getting your Linux IP Masquerading system working in the shortest time possible. David only plays a technical writer on T.V. so you might find the information in this document not as general and/or objective as it could be. If you think a section could be clearer, etc.. please let David know. The latest version of the MASQ HOWTO can be found at Dranch's Linux Page. Additional news, mirrors of the HOWTO, and information regarding IPMASQ can be found at the IP Masquerade Resource web page. If you have any technical questions on IP Masquerade, please join the IP Masquerade Mailing List instead of sending email to David or Ambrose. Most MASQ problems are -common- for ALL MASQ users and can be easily solved by users on the list. In addition to this, the response time of the IP MASQ email list will be much faster than a reply from either David or Ambrose.

The latest version of this document can be found at the following sites which also contains HTML, Postscript, PDF, etc. versions

- Dranch's Linux page
- http://ipmasq.webhop.net/: The IP Masquerade Resources
- http://ipmasq2.webhop.net/: The IP Masquerade Resources MIRROR
- The Linux Documentation Project
- Also refer to IP Masquerade Resource Mirror Sites Listing for other local mirrored sites.

1.3. Copyright & Disclaimer

The information herein this document is, to the best of David's knowledge, correct. However, the Linux IP Masquerade feature is written by humans and thus, the chance of mistakes, bugs, etc. might occur from time to time.

No person, group, or other body is responsible for any damage on your computer(s) and any other losses by using the information on this document. i.e.

"*THE AUTHORS AND ALL MAINTAINERS ARE NOT RESPONSIBLE FOR ANY DAMAGES INCURRED DUE TO ACTIONS TAKEN BASED ON THE INFORMATION IN THIS DOCUMENT.* "

Ok, with all this behind us... On with the show..

Chapter 2. Background Knowledge

2.1. What is IP Masquerade?

IP Masquerade is a networking function in Linux similar to the one-to-many (1:Many) NAT (Network Address Translation) servers found in many commercial firewalls and network routers. For example, if a Linux host is connected to the Internet via PPP, Ethernet, etc., the IP Masquerade feature allows other "internal" computers connected to this Linux box (via PPP, Ethernet, etc.) to also reach the Internet as well. Linux IP Masquerading allows for this functionality even though these internal machines don't have **an officially assigned IP address**.

MASQ allows a set of machines to **invisibly** access the Internet via the MASQ gateway. To other machines on the Internet, the outgoing traffic will appear to be from the IP MASQ Linux server itself. In addition to the added functionality, IP Masquerade provides the foundation to create a HEAVILY secured networking environment. With a well built firewall, breaking the security of a well configured masquerading system and internal LAN should be considerably difficult to accomplish.

If you would like to know more on how MASQ (1:Many) differs from 1:1 (true) NAT and Proxy solutions, please see the Section 7.6 FAQ entry.

2.2. Current Status

IP Masquerade has been in the Linux kernels for several years now and is quite mature as the kernel enters the 2.4.x stage. Kernels since Linux 1.3.x have had MASQ support built-in. Today, many individuals and commercial businesses are using it with excellent results.

2.4.x kernel users:

- The 2.4.x kernel hosts an entirely re-written set of NAT code which is both far superior, faster, and more secure than any previous versions written for Linux. Unfortunately, several kernel modules that were written for the 2.2.x kernel to support things like UDP-based RealAudio, etc. have not been ported to 2.4.x yet. Because of this, some people should consider NOT upgrading if these network applications are critical to them. But, at the same time, some of these programs have been updated and now use different, NAT-friendly protocols. Thus special NAT treatment is no longer required. As always, please see the http://ipmasq.webhop.net/: The IP Masquerade Resources site for updated news, etc.

Common network functionalities like Web browsing, telnet, ssh, ping, traceroute, etc. work well over stock IP Masquerade setups. Other network applications such as ftp, irc, and Real Audio work well with the appropriate additional IP MASQ modules loaded into the kernel as modules. Other network-specific programs like streaming audio (MP3s, True Speech, etc) should work too without any special module. Some users on the mailing list also had good results with video conferencing software.

It should be noted that running IP Masquerade with only ONE network card (NIC) to MASQ between internal and external Ethernet networks is NOT recommended. For more details, please see Section 7.28 FAQ section.

Anyways, please refer to Section 6.3 for a more complete listing of software supported by IP Maquerade all kernel versions.

IP Masquerade works well as a server to other 'client machines' running various operating systems and hardware platforms. Here is a sampling of successful reports with internal MASQed systems running :

- UNIX: Sun Solaris, [Net,Free,Open,*i]-BSD, Hp-UX, Linux, IBM AIX, Digital UNIX, Ultrix, etc.
- Microsoft Windows 2000, NT (3.x and 4.x), 95/98/ME, Windows for Workgroups (with the TCP/IP package)
- IBM OS/2
- Apple Macintosh MacOS machines running either MacTCP or Open Transport
- DOS-based systems with packet drivers and the NCSA Telnet package
- VAXen
- Compaq/Digital Alpha running Linux and NT
- Amiga computers with AmiTCP or AS225-stack.

The list goes on and on but the point is, if your OS platform talks TCP/IP, it should work with Linux's IP Masquerade!

2.3. Who Can Benefit From IP Masquerade?

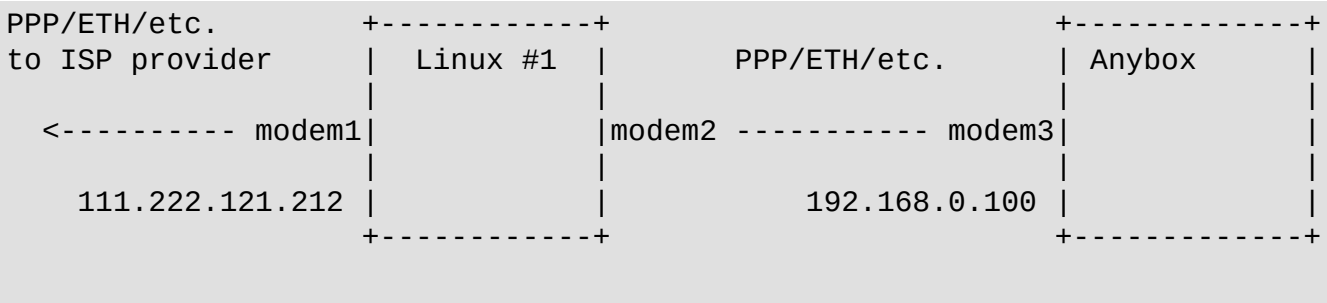
- If you have a Linux host connected to the Internet and..
- if you have internal computers running TCP/IP connected that are connected to this Linux box via on a network, and/or
- if your Linux host has more than one modem and acts as a PPP or SLIP server connected to *other* computers, and these machines do not have official or public assigned IP addresses (i.e. addressed with private TCP/IP numbers).
- If you want those *OTHER* machines to communicate to the Internet without spending extra money to acquire additional Public / Official TCP/IP addresses from your ISP, then you should either configure Linux to be a router or purchase an external router.

2.4. Who Doesn't Need IP Masquerade?

- If your machine is a stand-alone Linux host connected to the Internet (setting up a firewall is a good idea though), or
- if you already have multiple assigned public addresses for your *OTHER* machines, and
- if you don't like the idea of a 'free ride' using Linux and feel more comfortable using expensive commercial tools to perform the exact same functionalities.

2.5. How does IP Masquerade Work?

Based from the original IP Masquerade FAQ by Ken Eves: Here is a drawing of the most simplistic setup:



In the above drawing, a Linux box with IP_MASQUERADING is installed as Linux #1 and is connected to the Internet via PPP, Ethernet, etc. It has an assigned public IP address of 111.222.121.212. It also has another network interface (e.g. modem2) connected to allow incoming network traffic be it from a PPP connection, Ethernet connection, etc.

The second system (which does not need to be Linux) connects into the Linux #1 box and starts its network traffic to the Internet. This second machine does NOT have a publicly assigned IP address from the Internet, so it uses an RFC1918 private address, say 192.168.0.100. (see below for more info)

With IP Masquerade and the routing configured properly, this second machine "Anybox" can interact with the Internet as if it was directly connected to the Internet with a few small exceptions [noted later].

Quoting Pauline Middelink (the founder of Linux's IPMASQ):

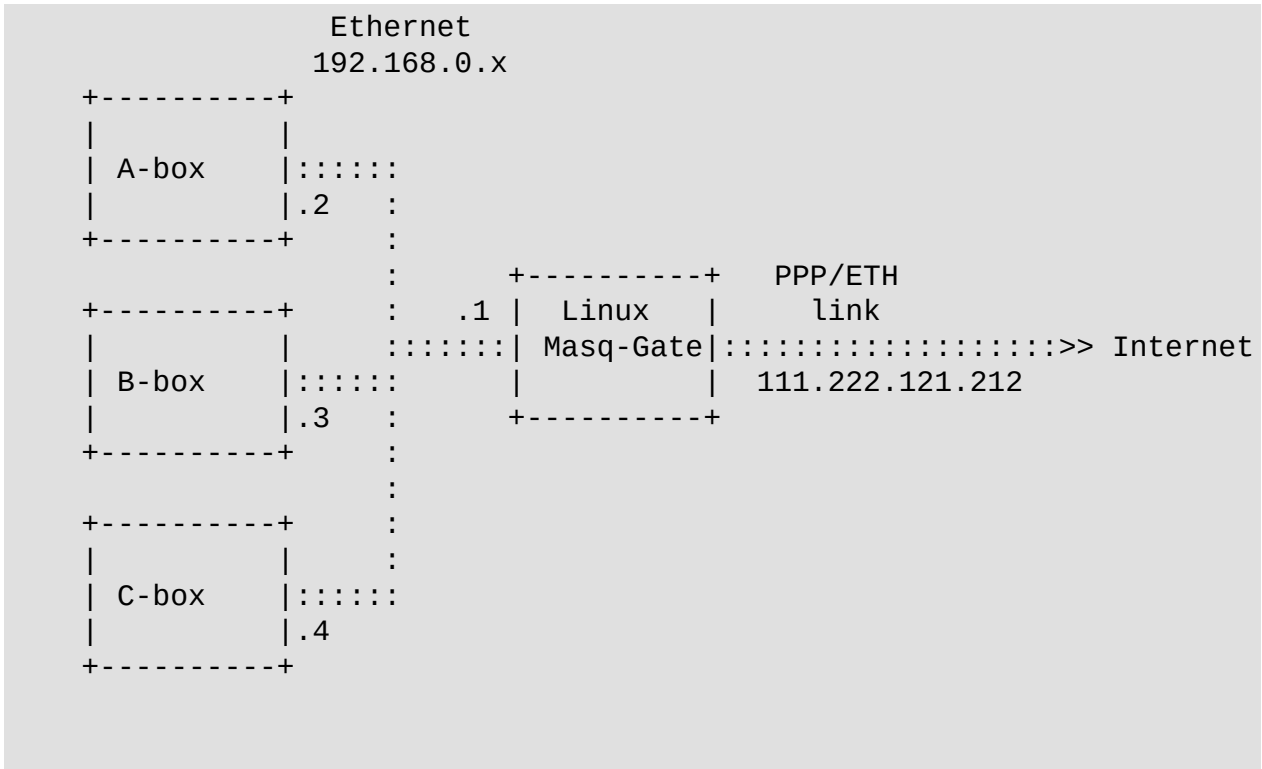
"Do not forget to mention that the "ANYBOX" machine should have the Linux #1 box configured as its default gateway (whether it be the default route or just a subnet is no matter). If the "ANYBOX" machine is connected via a PPP or SLIP connection, the Linux #1 machine should be configured to support proxy arp for all routed addresses. But, the setup and configuration of proxy arp is beyond the scope of this document. Please see the PPP-HOWTO for more details."

The following is an excerpt on how IPMASQ briefly works though this will be explained in more detail later. This short text is based from a previous post on comp.os.linux.networking which has been edited to match the names used in the above example:

```
   o I tell machine ANYBOX that my PPP or Ethernet connected Linux box is
its
     gateway.

   o When a packet comes into the Linux box from ANYBOX, it will assign the
     packet to a new TCP/IP source port number and insert its own IP address
     inside the packet header, saving the originals.  The MASQ server will
     then send the modified packet over the PPP/ETH interface onto the
     Internet.

   o When a packet returns from the Internet into the Linux box, Linux
     examines if the port number is one of those ports that was assigned
     above.  If so, the MASQ server will then take the original port and
     IP address, put them back in the returned packet header, and send
     the packet to ANYBOX.

   o The host that sent the packet will never know the difference.
```

Another IP Masquerading Example:

A typical example is given in the diagram below:



```
    |                        |          |                              >
    | <-Internal Network--> |          | <- External Network ----> >
    |   connected via an    |          |    Connected from the     >
    |   Ethernet hub or     |          |    Linux server to your   >
    |       switch          |          |    Internet connection    >
```

In this example, there are (4) computer systems that we are concerned about. There is also presumably something on the far right that your PPP/ETH connection to the Internet comes through (modem server, DSL DSLAM, Cablemodem router, etc.). Out on the Internet, there exists some remote host (very far off to the right of the page) that you are interested in communicating with). The Linux system named *Masq-Gate* is the IP Masquerading gateway for ALL internal networked machines. In this example, the machines *A-box*, *B-box*, and *C-box* would have to go through the Masq-Gate to reach the Internet. The internal network uses one of several RFC-1918 assigned private network addresses, where in this case, would be the Class-C network 192.168.0.0. If you aren't familiar with RFC1918, it is encouraged to read the first few chapters of the RFC but the jist of it is that the TCP/IP addresses 10.0.0.0/8, 172.16-31.0.0/12, and 192.168.0.0/16 are reserved. When we say "reserved", we mean that anyone can use these addresses as long as they aren't routed over the Internet. ISPs are even allowed to use this private addressing space as long as they keep these addresses within their own networks and NOT advertise them to other ISPs. Unfortunately, this isn't always the case but thats beyond the scope of this HOWTO.

Anyway, the Linux box in the diagram above has the TCP/IP address 192.168.0.1 while the other systems has the addresses:

- A-Box: 192.168.0.2
- B-Box: 192.168.0.3
- C-Box: 192.168.0.4

The three machines, `A-box`, `B-box` and `C-box`, can have any one of several operating systems, just as long as they can speak TCP/IP. Some such as *Windows 95*, *Macintosh MacTCP or OpenTransport* , or even another *Linux box* have the ability to connect to other machines on the Internet. When running the IP Masquerade, the masquerading system or `MASQ-gate` converts all of these internal connections so that they appear to originate from the `masq-gate` itself. MASQ then arranges so that the data coming back to a masqueraded connection is relayed to the proper originating system. Therefore, the systems on the internal network are only able to see a direct route to the internet and are unaware that their data is being masqueraded. This is called a "Transparent" connection.

NOTE: Please see Chapter 7 for more details on topics such as:

- The differences between NAT, MASQ, and Proxy servers.
- How packet firewalls work

2.6. Requirements for IP Masquerade on Linux 2.4.x

" ** *Please refer to IP Masquerade Resource for the latest information.* ** "

- The newest 2.4.x kernels are now using both a completely new TCP/IP network stack as well as a new NAT sub-system called NetFilter. Within this NetFilter suite of tools, we now have a tool called IPTABLES for the 2.4.x kernels much like there was IPCHAINS for the 2.2.x kernels and IPFWADM for the 2.0.x kernels. The new IPTABLES system is far more powerful (combines several functions into one place like true NAT functionality), offers better security (stateful inspection), and better performance with the new 2.4.x TCP/IP stack. But this new suite of tools can be a bit complicated in comparison to older generation kernels. Hopefully, if you follow along with this HOWTO carefully, setting up IPMASQ won't be too bad. If you find anything unclear, downright wrong, etc. please email David about it.

  Unlike the migration to IPCHAINS from IPFWADM, the new NetFilter tool has kernel modules that can actually support

older IPCHAINS and IPFWADM rulesets with minimal changes. So re-writing your old MASQ or firewall ruleset scripts is not longer required. **BUT..** with the 2.4.x kernels, you cannot use the old 2.2.x MASQ modules like ip_masq_ftp, ip_masq_irc, etc. **AND** IPCHAINS is incompatible with the new IPTABLES modules like ip_conntrack_ftp, etc. So, what does this mean? It basically means that if you want to use IPMASQ or PORTFW functionality under a 2.4.x kernel, you shouldn't use IPCHAINS rules but IPTABLES ones instead. Please also keep in mind that there might be several benefits in performing a full ruleset re-write to take advantage of the newer IPTABLES features like stateful tracking, etc. but that is dependant upon how much time you have to migrate your old rulesets. Please see Section 7.40 for additional details.

Some new 2.4.x functionalities include the following:

PROs:

- Lots of new protocols modules like: amanda, eggdrop, ipsec, ipv6, portscan, pptp, quota, rsh, talk, and tftp
- TRUE 1:1 NAT functionality for those who have TCP/IP addresses and subnets to use (no more iproute2 commands)
- Stateful application level (FTP, IRC, etc.) and stateful protocol level (TCP/UDP/ICMP) network traffic inspection
- Built-in PORT Forwarding (no more ipmasqadm or ipportfw commands)
- The built-in PORTFW'ing support works for both external and internal traffic. This means that users that have PORTFW for external traffic and REDIR for internal port redirection do not need to use two tools any more!
- PORT Forwarding of FTP traffic to internal hosts is now completely supported and is handled in the conn_trak_ftp module
- Full Policy-Based routing features (source-based TCP/IP address routing)
- Compatibility with Linux's FastRoute feature for significantly faster packet forwarding (a.k.a Linux network switching).

  Note that this feature is still not compatible with packet filtering for strong firewall rulesets.
- Fully supports TCP/IP v4, v6, and even DECnet (ack!)
- Supports wildcard interface names like "ppp*" for serial interfaces like ppp0, ppp1, etc
- Supports filtering on both input and output INTERFACES (not just IP addresses)
- Source Ethernet MAC filtering
- Denial of Service (DoS) packet rate limiting
- Packet REJECTs now have user-selectable return ICMP messages
- Variable levels of logging (different packets can go to different SYSLOG levels)
- Other features like traffic mirroring, securing traffic per login, etc.

CONs:

- Netfilter is an entirely new architechure thus most of the older 2.2.x MASQ kernel modules written to make non-NAT friendly network applications work through IPMASQ need to be re-written for the 2.4.x kernels. Because of this, if you specifically need functionality from some of these modules (see below), you should stay with a 2.2.x kernel until these modules have been either ported or the application has been updated to use NAT-friendly protocols. If you are curious on the porting status of a given module, please email the author of the module and NOT David or Ambrose. We don't code.. we just document. :-)

  Here is the status of the known IP Masq kernel modules or patches as found on the IPMASQ WWW site's Application Support Matrix. In addition, you should also setup out the Netfilter Patch-o-Matic URL as well. If you have the time and knowledge to help in the porting of code, your efforts would be highly appreciated:

```
 Status   = Module name =      Description and notes
---------   -----------   ---------------------------------
 Ported     CuSeeme      Used for Video conferencing

NotPorted   DirectPlay    Used for online Microsoft-based games

 Ported        FTP        Used for file transfers
                          - NOTEs:  Built into the kernel and
                                    fully supports PORTFWed FTP

ReWritten     H.323       Used for Video conferencing

NotPorted      ICQ        Used for Instant messaging
                          * No longer required for modern ICQ clients

 Ported        Irc        Used for Online chat rooms

 Ported      Quake        Used for online Quake games

 Ported       PPTP        Allow for multiple clients to the same server

NotPorted   Real Audio    Used for Streaming video / audio
                          * No longer required for modern RealVideo clients

NotPorted    VDO Live     Used for Streaming audio?
```

Documentation on how to perform MASQ module porting is available at http://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.html. If you have the time and knowledge, your talent would highly be appreciated in porting these modules.

If you'd like to read up more on NetFilter and IPTables, please see: http://www.netfilter.org/documentation/index.html#HOWTO and more specifically http://www.netfilter.org/documentation/HOWTO//NAT-HOWTO.html

Linux 2.4.x IP Masquerade requirements include:

- Any decent computer hardware. See Section 7.2 for more details.
- The 2.4.x kernel source is available from http://www.kernel.org/.

  NOTE: Most modern Linux distributions, Section 7.1, that natively come with 2.4.x kernels are typically modular kernels and have all the IP Masquerade functionality already included. In such cases, there is no need to compile a new Linux kernel. If you are UPGRADING your kernel, you should be aware of other programs that might be required and/or need to be upgraded as well (mentioned later in this HOWTO).
- The program "iptables" version 1.2.4 or newer (1.2.7a or newer is highly recommended) archive available from http://www.netfilter.org/

  - NOTE #1: All versions of IPTABLES less than 1.2.3 have a FTP module issue that can bypass any existing firewall rulesets. ALL IPTABLES users are highly recommended to upgrade to the newest version. The URL is above.

    NOTE #2: All versions of IPTABLES less than 1.2.2 have a FTP "port" security vulnerability in the ip_conntrack_ftp module. All IPTABLES users are highly recommended to upgrade to the newest version. The URL is above.
  - This tool, much like the older IPCHAINS and IPFWADM tools enables the various Masquerding code, more advanced

forms of NAT, packet filtering, etc. It also makes use of additional MASQ modules like the FTP and IRC modules. Additional information on version requirements for the newest IPTABLES howto, etc. is located at the Unreliable IPTABLES HOWTOs page.

- Loadable kernel modules, preferably 2.1.121 or higher, are available from http://home.pi.se/blox/modutils/index.html or ftp://ftp.kernel.org/pub/linux/utils/kernel/modutils
- A properly configured and running TCP/IP network running on the Linux machine as covered in Linux NET HOWTO and the Network Administrator's Guide . Also check out the TrinityOS document which is also authored by David Ranch. TrinityOS is a very comprehensive guide for Linux networking. Some topics include IP MASQ, security, DNS, DHCP, Sendmail, PPP, Diald, NFS, IPSEC-based VPNs, and performance sections, to name a few. There are over Fifty sections in all!
- Connectivity to the Internet for your Linux host covered in Linux ISP Hookup HOWTO, Linux PPP HOWTO, and TrinityOS. Other helpful HOWTOs could include: Linux DHCP mini-HOWTO, Linux Cable Modem mini-HOWTO and http://www.tldp.org/HOWTO/DSL-HOWTO/index.html
- Know how to configure, compile, and install a new Linux kernel as described in the Linux Kernel HOWTO. This HOWTO does cover kernel compiling but only for IP Masquerade related options.

2.7. Requirements for IP Masquerade on Linux 2.2.x

" ** *Please refer to IP Masquerade Resource for the latest information.* ** "

- Any decent computer hardware. See Section 7.2 for more details.
- The 2.2.x kernel source is available from http://www.kernel.org/.

  NOTE: Most modern Linux distributions, Section 7.1, that natively come with 2.2.x kernels are typically modular kernels and have all the IP Masquerade functionality already included. In such cases, there is no need to compile a new Linux kernel. If you are UPGRADING your kernel, you should be aware of other programs that might be required and/or need to be upgraded as well (mentioned later in this HOWTO).

  - NOTE #1: --- UPDATE YOUR KERNEL --- Linux 2.2.x kernels less than version 2.2.20 contain several different security vulnerabilities (some were MASQ specific). Kernels less than 2.2.20 have a few local vulnerabilities. Kernel versions less than 2.2.16 have a TCP root exploit vulnerability and versions less than 2.2.11 have a IPCHAINS fragmentation bug. Because of these issues, users running a firewall with strong IPCHAINS rulesets are open to possible instrusion. Please upgrade your kernel to a fixed version.
- NOTE #2: Some newer Section 7.1 such as Redhat 5.2 might not be Linux 2.2.x ready (upgradable). Tools like DHCP, NetUtils, etc. will need to be upgraded. More details can be found later in the HOWTO.
- Loadable kernel modules, preferably 2.1.121 or higher, are available from http://home.pi.se/blox/modutils/index.html or ftp://ftp.kernel.org/pub/linux/utils/kernel/modutils
- A properly configured and running TCP/IP network running on the Linux machine as covered in Linux NET HOWTO and the Network Administrator's Guide . Also check out the TrinityOS document which is also authored by David Ranch. TrinityOS is a very comprehensive guide for Linux networking. Some topics include IP MASQ, security, DNS, DHCP, Sendmail, PPP, Diald, NFS, IPSEC-based VPNs, and performance sections, to name a few. There are over Fifty sections in all!
- Connectivity to the Internet for your Linux host covered in Linux ISP Hookup HOWTO, Linux PPP HOWTO, and TrinityOS. Other helpful HOWTOs could include: Linux DHCP mini-HOWTO, Linux Cable Modem mini-HOWTO and http://www.tldp.org/HOWTO/DSL-HOWTO/index.html
- IP Chains 1.3.10 or newer are available from http://www.netfilter.org/ipchains/. Additional information on version requirements for the newest IPCHAINS HOWTO, etc is located at the Linux IP Chains page (mirror at Samba.org)

- Know how to configure, compile, and install a new Linux kernel as described in the Linux Kernel HOWTO. This HOWTO does cover kernel compiling but only for IP Masquerade related options.

Other optional patches and tools for 2.2.x kernels

- TCP/IP port-forwarding or re-directing:
  - IP PortForwarding (IPMASQADM) - RECOMMENDED - mirror
- PORTFW FTP Solutions:
  - There are 2.2.x and 2.0.x kernel MASQ Module solutions for PORTFWed FTP to a MASQed machine (put an FTP server behind a MASQ server). Please see the Application Page on the IPMASQ WWW site for full details. Please note that this is not required for 2.4.x kernels.

    There is a full FTP proxy application from SuSe that will also allow PORTFWed-like functionality to reach an internal FTP server. For more details, please refer to the SuSe Proxy URL.
- IPROUTE2 for True 1:1 NAT, Policy-based (source) routing, and Traffic Shaping:
  - ftp://ftp.inr.ac.ru/ip-routing
  - Documentation can be found at http://www.compendium.com.ar/policy-routing.txt
  - The Advanced Routing HOWTO
  - Some source code mirrors are at:

    ftp://ftp.funet.fi/pub/mirrors/ftp.inr.ac.ru/ip-routing/ (STM1 to USA) --- ftp://sunsite.icm.edu.pl/pub/Linux/iproute/

    ftp://ftp.sunet.se/pub/Linux/ip-routing/ --- ftp://ftp.nvg.ntnu.no/pub/linux/ip-routing/

    ftp://ftp.crc.ca/pub/systems/linux/ip-routing/ --- ftp://ftp.paname.org (France)

Please see the IP Masquerade Resource page for more information available on these patches and possibly others as well.

2.8. Requirements for IP Masquerade on Linux 2.0.x

" ** *Please refer to IP Masquerade Resource for the latest information.* ** "

- Any decent computer hardware. See Section 7.2 for more details.
- The 2.0.x kernel source is available from http://www.kernel.org/.

  NOTE: Most modern Linux Section 7.1 that natively come with 2.0.x kernels are typically modular kernels and have all the IP Masquerade functionality already included. In such cases, there is no need to compile a new Linux kernel. If you are UPGRADING your kernel, you should be aware of other programs that might be required and/or need to be upgraded as well (mentioned later in this HOWTO).
- Loadable kernel modules, preferably 2.1.85 or newer is available from http://home.pi.se/blox/modutils/index.html or ftp://ftp.kernel.org/pub/linux/utils/kernel/modutils (modules-1.3.57 is the minimal requirement)
- A properly configured and running TCP/IP network running on the Linux machine as covered in Linux NET HOWTO and the Network Administrator's GuideAlso check out the TrinityOS document which is also authored by David Ranch.

TrinityOS is a very comprehensive guide to Linux networking. Topics include IP MASQ, security, DNS, DHCP, Sendmail, PPP, Diald, NFS, IPSEC-based VPNs, performance issues, and many more. There exists over fifty sections in all!

- Connectivity to the Internet for your Linux host is covered in Linux ISP Hookup HOWTO, Linux PPP HOWTO, and TrinityOS. Other helpful HOWTOs could include: Linux DHCP mini-HOWTO, Linux Cable Modem mini-HOWTO and Linux DSL HOWTO
- Ipfwadm 2.3.0 or newer is available from http://www.xos.nl/linux/ipfwadm/download.html
- More information on version requirements are on the Linux IPFWADM page
- If you are interested in running IPCHAINS on a 2.0.x+ kernel, see Willy Tarreau's IPCHAINS enabler for 2.0.36+ or Rusty's IPCHAINS for 2.0.x kernels. Please note that these patches are NOT compatible with the IPPORTFW patches for the 2.0.x kernels. Unfortunately, its an either/or deal.
- Know how to configure, compile, and install a new Linux kernel as described in the Linux Kernel HOWTO. This HOWTO does cover kernel compiling but only for IP Masquerade related options.

Here is a list of IP Masquerading patches for 2.0.x kernels:

- Steven Clarke's IP PortForwarding (IPPORTFW) - *RECOMMENDED*
- IP AutoForward - NOT Recommended
- REDIR for TCP (REDIR) - NOT Recommended unless required for internal PORTFW
- UDP redirector (UDPRED) - NOT Recommended
- PORTFWed FTP:
    - If you are going to port forward FTP traffic to an internal FTP server, you might need to download Fred Viles's FTP server patch The reason for "might" is that some users have had success without the use of these pathches, while others need it. Explicit details on this topic can be found in Section 6.7 of this HOWTO.
- X-Windows display forwarders:
    - X-windows forwarding (DXCP)
- PPTP (GRE) and SWAN (IPSEC) VPNs tunneling forwarders:
    - If you plan connecting an internal MASQed PC to a remote PPTP server, you MUST INSTALL the PPTP-Masquerade kernel patch available from the URLsbelow. If you plan on having external PPTP users connect to an internal masqueraded PPTP server, not only do you need the kernel patch installed but you also need PORTFW support enabled in the kernel. Please see the following URLs for the patches and more information:

      John Hardin's VPN Masquerade forwarders or the old patch for just PPTP Support.
- Game specific patches:
    - Glenn Lamb's LooseUDP for 2.0.36+ patch.

Chapter 3. Setting Up IP Masquerade

3.1. Compiling a new kernel if needed

If your private network contains any vital information, think carefully in terms of SECURITY before implementing IP Masquerade.

By default, IP MASQ becomes a GATEWAY for you to get onto the Internet, but it also can allow someone from the Internet to possibly get into your internal network.

Once you have IP MASQ functioning, it is HIGHLY recommended for the user to implement a STRONG IPFWADM/ IPCHAINS firewall ruleset. Please see Section 6.4.1, Section 6.4.2 and Section 6.4.3 located below for more details.

3.2. Checking your existing kernel for MASQ functionality

Almost ALL modern Linux distributions come MASQ-Ready these days but its always good to check your system before you try to set things up. Follow these few steps for your kernel to see if your kernel is MASQ ready.

To see which kernel your system is running, run the following command:

```
uname -a
```

- Just for clarity: 2.4.x kernels run IPTABLES :: 2.2.x kernels run IPCHAINS :: 2.0.x kernels run IPFWADM
- In general, you must have kernel support for:
    - IP forwarding
    - IP masquerading
    - IP Firewalling
    - etc.

You will also need to have most MASQ-related modules compiled (most modular kernels will already have all you need already done. Then you will NOT need to re-compile the kernel. If you AREN'T SURE if your Linux distribution is MASQ ready, do the following:

- **2.4.x kernels** (look for most of the following entries out of the much longer list):
    - Run the command "`ls /proc/sys/net/ipv4`" while logged into the Linux box. These items are required and should be present regardless if your kernel built IPMASQ as modules or statically.
        - `ip_dynaddr`

          `ip_forward`
    - To check if IPMASQ was compiled statically into the kernel, run the command "`/sbin/lsmod`" and see if and modules like the ones shown below for the MODULE section are loaded. No? Ok, now run the command "`ls /proc/net/`" and see if you see additional /proc files such as:
        - `ip_masquerade`

          `ip_conntrack`

          `ip_tables_names`

      If you see these /proc entries and there WEREN'T any kernel modules loaded (shown via the "lsmod" command mentioned above), then your kernel has the IPTABLES subsystem statically compiled into it and is ready to go to use IPMASQ on this system.

- If your kernel uses IPTABLES via modules, most of the stuff listed above should have been missing (because the modules probably aren't loaded). Run the command "`ls /lib/modules/`uname -r`/kernel/net/ipv4/netfilter/`" where you should see files like:

  - `ip_conntrack.o, ip_conntrack_ftp.o, ip_conntrack_irc.o, ip_nat_ftp.o, ip_nat_irc.o`

    `ip_tables.o, ipt_MASQUERADE.o, iptable_nat.o, iptable_mangle.o, iptable_filter.o`

    And some optional ones like: `ipchains.o, ipt_REJECT.o, and ipt_tcpmss.o`

  If you see those kernel files, IPTABLES was compiled using modules and things look ready to go to use IPMASQ on this system.

- **2.2.x kernels** (look for most of the following entries out of the much longer list): list):

  - Run the command "`ls /proc/sys/net/ipv4`" while logged into the Linux box. These items are required and should be present regardless if your kernel built IPMASQ as modules or statically.

    - `ip_always_defrag`

      `ip_dynaddr`

      `ip_forward`

      `ip_masq_debug`

      `ip_masq_udp_dloose` (some distros don't support this -- ignore it for now

      Other 2.2.x options can be checked by running "ls /proc/net/"

      - `ip_fwchains`

        `ip_fwnames`

        `ip_masquerade`

      Even more 2.2.x options can be checked by running "ls /proc/net/"

      - `app`

        `icmp`

        `icq`

        `mfw`

        `portfw`

        `tcp`

```
udp/
```

- **2.0.x kernels** (look for most of the following entries out of the much longer list):
  - Run the command "`ls /proc/sys/net/ipv4`" while logged into the Linux box. These items are required and should be present regardless if your kernel built IPMASQ as modules or statically.
    - `ip_dynaddr`

      `ip_forward`

      running "ls /proc/net"
      - `ip_forward`

        `ip_masq_app`

        `ip_masquerade`

        `ip_portfw`

Ultimately, it comes down to the fact if you see /proc files such as "`ip_forward`", "`ip_masq_debug`", "`ip_masq_udp_dloose`"(optional), and "`ip_always_defrag`" (optional) exist.

So. Do most of the above /proc entries or kernel modules show up for your respective kernel? If so, thats good! If you cannot find any of the above entries or if you aren't sure if your distribution supports IP Masquerading by default, ASSUME IT DOESN'T SUPPORT MASQ. You can do one last check by looking at the Section 7.1 section and see if your Linux Distribution is listed. Still not there? Sounds like you'll need to compile a kernel but don't worry.. it isn't hard.

Regardless if your current kernel has MASQ support or not, reading the remainder of this section is still highly recommended as it contains other useful information.

3.2.1. Compiling Linux 2.4.x Kernels

- First, you'll need to get some 2.4.x kernel sources (preferably the latest kernel version - NEWER *IS* BETTER IN LINUX LAND)
  - NOTE #1: As both the 2.4.x kernel train and the iptables program development progresses, the compile configurion options will change over time. As of this version of the IPMASQ howto, this section reflects the settings for IPTABLES 1.2.7a and the 2.4.20 kernel. If you are compiling against a newer or previous kernel or IPTABLES version, the dialogs and even commands might look different. It is recommended that you update to the newest versions of both the kernel and IPTABLES for added capability, performance, and stability of the kernel.
- Next, depending on the version of the Linux kernel and IPTABLES archive you downloaded, you **might** want to apply some IPTABLES "patch-o-matic" patches against the kernel. These OPTIONAL patches might fix some known problems, add additional functionality you might need (H.323 protocol, specific issues with network games), etc. It should be noted that the Patch-O-Matic patches used to come with the IPTABLES archive. This is no longer the case and you have to download them (if any) seperately. You can find the the various URLs for downloading IPTABLES, the Patch-o-matic system, etc. Section 2.6.
- If this is your first time compiling the kernel, don't be scared. In fact, it's rather easy and it's covered in several URLs found in Section 2.6. Please note that the instructions included here is just one way to do build a kernel. Please see the Kernel HOWTO

for full details.

NOTE: Please notice that it **IS NOT** recommended to put the new kernel sources into the /usr/src/linux directory. You should leave the original kernel sources that came with your Linux distribution in /usr/src/linux. For more details on this topic, please read the "README" file in the top level directory of the kernel sources.

- For this HOWTO example, create a directory called `/usr/src/kernel`. Next, "cd" into this directory and download the newest 2.4.x kernel sources into it. Once downloaded, issue the following command (if the file ends in a .tar.gz): `tar xvzf linux-2.4.x.tar.gz` or (if the file ends in a .tar.bzip2): `tar xyvf linux-2.4.x.tar.bz2`. Please substitute the "x" in the 2.4.x filename with the Linux 2.4 kernel version you downloaded.

  BZ2 Note: Some Linux distributions use the "I" option instead of the "y" option to decompress bzip2 archives.

  Once uncompressed, I recommend that you rename the directory from the stock "linux" name to "linux-2.4.x" (replace the "x" with the specific version of your newly installed kernel) for clarity. To do this, run the command "`mv linux linux-2.4.x`". Next, make sure there is a directory or symbolic link pointing to "`/usr/src/kernel/linux`" ie. run the command:

```
ln -s /usr/src/kernel/linux-2.4.x /usr/src/kernel/linux
```

  again subsituting the "x" for your proper kernel version.

- As mentioned above, you might consider applying any appropriate or optional patches to the kernel's MASQ code BEFORE you compile the final kernel. The IP MASQ code found in the stock kernels is already very useful and does not require any specific patching in order for the system to work for NAT-friendly network applications. Many of these patches are only to fix possible known bugs, add new features (some are /very/ cool), etc. Please refer to Section 2.6 for URLs and the IP Masquerade Resources for up-to-date information and patch URLs.
- **Applying IPTABLES and Patch-o-Matic kernel patches**

  Download the iptables package and optional Patch-O-matics from the Section 2.6 and put it into a directory, say "`/usr/src/archive/netfilter`". Next, go into this new netfilter directory and uncompress the iptables archive with the command:

```
tar xyvf iptables-x.y.z.tar.bz2
tar xyvf patch-o-matic-x.tar.bz2
```

  Now, go into the new iptables-x.y.x directory (/usr/src/archive/netfilter/iptables-x.y.z) and run the command

```
#For iptables v1.2.7a:
make KERNEL_DIR=/usr/src/kernel/linux

#For iptables v1.2.4 (when Patch-o-matic was built-in):
make pending-patches KERNEL_DIR=/usr/src/kernel/linux
```

  NOTE: this assumes that your 2.4.x kernel sources are in the `/usr/src/kernel/linux` directory.

  NOTE #2: If you append a "/" to the end of the above command line, you will get an error stating:

```
"make: *** [/usr/src/kernel/linux/include/asm/socket.h] Error 1".
```

Remove the trailing "/" and try again.

Here is an example of compiling IPTABLES v1.2.7a. Your output might look different depending on what version you are trying to use.

```
# make KERNEL_DIR=/usr/src/kernel/linux

Extensions found:

cc -O2 -Wall -Wunused -I/usr/src/kernel/linux/include -Iinclude/
-DIPTABLES_VERSION=\"1.2.7a\"  -fPIC -o extensions/libipt_ah_sh.o -c
extensions/libipt_ah.c
ld -shared -o extensions/libipt_ah.so extensions/libipt_ah_sh.o
cc -O2 -Wall -Wunused -I/usr/src/kernel/linux/include -Iinclude/
-DIPTABLES_VERSION=\"1.2.7a\"  -fPIC -o extensions/libipt_conntrack_sh.o -c
extensions/libipt_conntrack.c
ld -shared -o extensions/libipt_conntrack.so extensions/libipt_conntrack_sh.o
cc -O2 -Wall -Wunused -I/usr/src/kernel/linux/include -Iinclude/
-DIPTABLES_VERSION=\"1.2.7a\"  -fPIC -o extensions/libipt_dscp_sh.o -c
extensions/libipt_dscp.c
extensions/libipt_dscp_helper.c:69: warning: `dscp_to_name' defined but not
used
ld -shared -o extensions/libipt_dscp.so extensions/libipt_dscp_sh.o
.
.
.
cc -O2 -Wall -Wunused -I/usr/src/kernel/linux/include -Iinclude/
-DIPTABLES_VERSION=\"1.2.7a\"    -c -o libipulog/libipulog.o
libipulog/libipulog.c
ar rv libipulog/libipulog.a libipulog/libipulog.o
a - libipulog/libipulog.o
rm libiptc/libip6tc.o libiptc/libip4tc.o libipulog/libipulog.o libipq/libipq.o
```

- Ok, hopefully the IPTABLES program compiled up for you. Now, you need to install it. To do this, directory and run the command

```
make install KERNEL_DIR=/usr/src/kernel/linux
```

- Here is an example of installing IPTABLES v1.2.7a. Your output might look different depending on what version you are trying to use.

```
# make install KERNEL_DIR=/usr/src/kernel/linux

cp iptables /usr/local/sbin/iptables
cp iptables-save /usr/local/sbin/iptables-save
cp iptables-restore /usr/local/sbin/iptables-restore
cp ip6tables /usr/local/sbin/ip6tables
cp extensions/libipt_ah.so /usr/local/lib/iptables/libipt_ah.so
cp extensions/libipt_conntrack.so /usr/local/lib/iptables/libipt_conntrack.so
cp extensions/libipt_dscp.so /usr/local/lib/iptables/libipt_dscp.so
cp extensions/libipt_ecn.so /usr/local/lib/iptables/libipt_ecn.so
cp extensions/libipt_esp.so /usr/local/lib/iptables/libipt_esp.so
cp extensions/libipt_helper.so /usr/local/lib/iptables/libipt_helper.so
.
.
.
cp extensions/libip6t_udp.so /usr/local/lib/iptables/libip6t_udp.so
cp extensions/libip6t_LOG.so /usr/local/lib/iptables/libip6t_LOG.so
cp extensions/libip6t_MARK.so /usr/local/lib/iptables/libip6t_MARK.so
```

Next, if you are interested in applying a Patch-O-Matic patch set, go into the `patch-o-matic-X` directory (/usr/src/archive/netfilter/patch-o-matic-X) and run the command

-

```
#For Patch-O-Matic later than the release of iptables v1.2.7a:
KERNEL_DIR=/usr/src/kernel/linux
./runme pending
```

  NOTE #1: The use of the "pending" batch is the most common for IPMASQ functionality but there are several others. See below.

  NOTE #2: this assumes that your 2.4.x kernel sources are in the `/usr/src/kernel/linux` directory.

  NOTE #3: If you append a "/" to the end of the command line, you will get an error stating:

```
"make: *** [/usr/src/kernel/linux/include/asm/socket.h] Error 1".
Remove the trailing "/" and try again.
```

  Here is an example of the Patch-O-Matic prompts you might receive for a 2.4.20 kernel with the "20030107" Patch-O-Matic set. You can also run the "runme" program in a batch mode to speed things up, add experimental patches, etc. if you'd like. To better understand your options, simply run the "`./runme`" command by itself. Please note that these prompts WILL CHANGE over time.

-

```
Welcome to Rusty's Patch-o-matic!

Each patch is a new feature: many have minimal impact, some do not.
Almost every one has bugs, so I don't recommend applying them all!
-------------------------------------------------------
Already applied: submitted/01_2.4.19
                 submitted/02_2.4.20
                 submitted/ipt_ULOG-mac_len-fix
                 submitted/ipt_multiport-invfix
                 pending/01_ip_conntrack_proto_tcp-lockfix
                 pending/02_newnat-udp-helper
                 pending/03_REJECT-fwspotting-phrack60-fix
                 pending/04_ftp-conntrack-msg-fix

Testing... 05_ECN-tcpchecksum-littleendian-fix.patch NOT APPLIED (1 rejects out
of 1 hunks)
The pending/05_ECN-tcpchecksum-littleendian-fix patch:
   Author: Patrick McHardy
   Status: Pending for kernel inclusion

   The 2.4.20 kernel included the new iptables 'ECN' target, enabling a
selective
   ECN disable mechanism.   Unfortunately there was a bug in the incremental
TCP
   checksum update, resulting in broken TCP checksums on little endian
machines.

   This patch fixes the Bug.

Testing patch pending/05_ECN-tcpchecksum-littleendian-fix.patch...
   Patch pending/05_ECN-tcpchecksum-littleendian-fix.patch applied cleanly.
Applying patch pending/05_ECN-tcpchecksum-littleendian-fix.patch...
   Patch pending/05_ECN-tcpchecksum-littleendian-fix.patch applied cleanly.

Excellent! Kernel is now ready for compilation.
```

- If everything patches fine, you should see something like the text

```
Excellent! Kernel is now ready for compilation.
```

  towards the bottom of the screen. Beyond that, you don't have to install anything at this point. The next step is to compile the new PATCHED kernel.
- Ok, now the new kernel is ready to be compiled but you should make sure that you also have the proper matching `iptables` program on your machine too (just to make sure). Run the command:

  -

    ```
    whereis iptables
    ```

  and make sure its installed on the machine (the default place is in `/usr/local/sbin/iptables`. If you cannot find it or

patched up your kernel sources as shown above, I recommend you just re-compile it up as shown above.

Now that the kernel sources are patched up, you need to configure it to know what kinds of features you need (HD support, Networking support, MASQ support, etc.). Here are the MINIMUM kernel configuration options required to enable IP Masquerade functionality. Please understand that this HOWTO illustrates just ONE way to configure and compile a kernel (modules vs static). The main difference from this example vs. an example given by a different MASQ guide is that some people might wish to compile kernel components either as **modules OR monolithically** into the kernel. Basically, compiling things as modules gives you added flexibility to what is or isn't installed into the kernel (reduces unneeded memory use for things you aren't / won't use and modules also allow for drop-in software upgrades [usually no need to reboot the machine]). On the flip side, kernel modules add more complexity to your configuration and sometimes the kernel auto-loader might make mistakes (not that I've ever seen this happen). Compiling things directly into the kernel makes things simpler BUT you loose a huge level of flexibility. The following kernel configuration example is a mixture of both a selection of kernel modules and building them in monolithically (you probably will ALWAYS need MASQ functionality ready to go).

- Side Note: It is assumed that you will also configure the kernel to use your other installed hardware such as USB printers, Ethernet network interfaces, SCSI and IDE HD controllers, etc. as well. Please refer to the Linux Kernel HOWTO and the kernel source's "`README`" file and "`Documentation/`" directory for detailed help on compiling a kernel.

You will need to answer either **YES, NO, or MODULE** to the following program. Not all options will be available without the proper kernel patches described later in this HOWTO. This shouldn't be an issue as most 3rd party patches are only needed for a very select group of users.

Run the following commands to configure your kernel:

- `cd /usr/src/kernel/linux`
- `make menuconfig`

Please note the following kernel prompts reflect a 2.4.14 kernel (with some of the optional Patch-O-Matic additions. Please read the following carefully for recommendations:

```
[ Code maturity level options ]

  * Prompt for development and/or incomplete code/drivers (CONFIG_EXPERIMENTAL) [Y/
n/?]
    - YES: though not required for IP MASQ, this option allows the kernel to create
           the MASQ modules and enable the option for port forwarding

  * Enable loadable module support (CONFIG_MODULES) [Y/n/?]
    - YES: allows you to load kernel IP MASQ modules

  * Set version information on all module symbols (CONFIG_MODVERSIONS) [Y/n/?]
    - YES: allows newer kernels to load older modules if possible

  * Kernel module loader (CONFIG_KMOD) [Y/n/?]
    - OPTIONAL: Recommended : allows the kernel to load various kernel modules as it
needs them

  == Non-MASQ options skipped
  ==   (CPU type, memory, SMP, FPU, specific stuff)


[ General setup ]

  * Networking support (CONFIG_NET) [Y/n/?]
```

```
    - YES: Enables the network subsystem

  == Non-MASQ options skipped
  ==   (specific hardware, PCI, kernel binaries, PCMCIA, etc.)


  * Sysctl support (CONFIG_SYSCTL) [Y/n/?]
    - YES:  Enables the ability to enable disable options such as forwarding,
      dynamic IPs, etc. via the /proc interface


[ Block devices ]

  == Non-MASQ options skipped
  ==   (kernel binaries, power management, PnP, RAID, etc.)

    == Don't forget to compile in support for hardware that you might need:
    ==   IDE controllers, HDs, CDROMs, etc.

[ Networking options ]

  * Packet socket (CONFIG_PACKET) [Y/m/n/?]
    - YES: Though this is OPTIONAL, this recommended feature will allow you
           to use TCPDUMP to debug any problems with IP MASQ

  * Packet socket: mmapped IO (CONFIG_PACKET_MMAP) [N/y/?] y
    - YES: Speed up the packet protocol

  * Kernel/User netlink socket (CONFIG_NETLINK) [Y/n/?]
    - OPTIONAL:  Recommended : this feature will allow the logging of
           advanced firewall issues such as routing messages, etc

  * Routing messages (CONFIG_RTNETLINK) [N/y/?] (NEW) y
    - OPTIONAL: Allows for support of advanced kernel routing messages
                if you enabled the CONFIG_NETLINK option

  * Netlink device emulation (CONFIG_NETLINK_DEV) [N/y/m/?] (NEW)
    - NO:  This option does not have anything to do with packet firewall
           logging

  * Network packet filtering (replaces ipchains) (CONFIG_NETFILTER) [N/y/?] y
    - YES: Enable this option to let IPTABLES configure the TCP/IP subsection
           of the kernel.  By enabling this, then you can turn on advanced
           routing mechanisms like IP Masq, packet filtering, etc.

  * Network packet filtering debugging (CONFIG_NETFILTER_DEBUG) [N/y/?] (NEW) n
    - NO: Not required for Masquerading functionality though it may help
          for troubleshooting.  There might be a performance penalty when
          enabling this.

  * Socket Filtering (CONFIG_FILTER) [Y/n/?]
    - OPTIONAL:  Recommended : Though this doesn't have anything do with IPMASQ,
      if you plan on implimenting a DHCP server on the internal network, you WILL
      need to enable this option.
```

```
  * Unix domain sockets (CONFIG_UNIX) [Y/m/n/?]
    - YES:  This enables the UNIX TCP/IP sockets mechanisms

  * TCP/IP networking (CONFIG_INET) [Y/n/?]
    - YES: Enables the TCP/IP protocol

  * IP: multicasting (CONFIG_IP_MULTICAST) [N/y/?]
    - OPTIONAL:  You can enable this if you want to be able to receive
                 Multicast traffic.  Please note that your ISP must
                 support Multicast as well for this all to work at all

  * IP: advanced router (CONFIG_IP_ADVANCED_ROUTER) [Y/n/?]
    - OPTIONAL:  Though there is nothing in this section mandatory for
                 Masquerade, some specific options might be useful

    == Non-MASQ options skipped
    ==   ( autoconf, tunneling )

  * IP: multicast routing (CONFIG_IP_MROUTE) [N/y/?] n
    - OPTIONAL:  Though not needed for IPMASQ, enabling this feature will
                 let you route multicast traffic through your Linux box.
                 Please note that this requires that your ISP be multicast
                 enabled as well.

    == Non-MASQ options skipped
    ==   (ARPd)

  * IP: TCP Explicit Congestion Notification support (CONFIG_INET_ECN) [N/y/?] n
    - NO: Though enabling this option would be great, there are many Internet
          sites out there that will block this.  Hit the "?" when configuring
          the kernel to learn more about it but it is recommended to say NO for
          now.

  * IP: TCP syncookie support (disabled per default) (CONFIG_SYN_COOKIES) [Y/n/?]
    - YES: Recommended : for basic TCP/IP network security


[ Networking options --> IP: Netfilter Configuration ]


  * Connection tracking (required for masq/NAT) (CONFIG_IP_NF_CONNTRACK) [N/y/m/?]
(NEW) m
    - YES: (Module) This enables the kernel to track various network connections.
           This option is required for Masquerading support as well as to enable
           Stateful tracking for various filewall mechanisms.  Please note that
           if you compile this directly into the kernel, you cannot enable
           the legacy IPCHAINS or IPFWADM compatibility modules.

  * FTP protocol support (CONFIG_IP_NF_FTP) [M/n/?] (NEW) m
    - YES: (Module) This enables the proper Masquerading of FTP connections if
           CONFIG_IP_NF_CONNTRACK was enabled above

  * IRC protocol support (CONFIG_IP_NF_IRC) [M/n/?] (NEW) m
    - YES: (Module) This enables the proper Masquerading of IRC connections if
           CONFIG_IP_NF_CONNTRACK was enabled above
```

```
  * Userspace queueing via NETLINK (EXPERIMENTAL) (CONFIG_IP_NF_QUEUE) [N/y/m/?]
(NEW) m
    - OPTIONAL: Though this is OPTIONAL, this feature will allow IPTABLES to
                copy specific packets to UserSpace tools for additional checks

  * IP tables support (required for filtering/masq/NAT) (CONFIG_IP_NF_IPTABLES) [N/y/
m/?] (NEW) m
    - YES: (Module) Enables IPTABLES support

  * limit match support (CONFIG_IP_NF_MATCH_LIMIT) [N/y/m/?] (NEW) y
    - OPTIONAL:  (Module) Recommended : Though not required, this option can used to
                 enable rate limiting of both traffic and loggin messages help slow
down denial
                 of service (DoS) attacks.

  * MAC address match support (CONFIG_IP_NF_MATCH_MAC) [N/y/m/?] (NEW) m
    - OPTIONAL:  Though not required, the option can allow you to
                 filter traffic based upon the SOURCE Ethernet MAC address.

  * netfilter MARK match support (CONFIG_IP_NF_MATCH_MARK) [N/y/m/?] (NEW) y
    - YES: (Module) Recommended : This enables IPTABLES to take action upon marked
packets.
           This mechanism can allow for PORTFW functionality, TOS marking, etc.

  * Multiple port match support (CONFIG_IP_NF_MATCH_MULTIPORT) [N/y/m/?] (NEW) y
    - YES: (Module) Recommended : This enables IPTABLES to accept mutliple SRC/DST
port
           ranges (non-contiguous) instead of one port range per IPTABLES
           statement.

  * TOS match support (CONFIG_IP_NF_MATCH_TOS) [Y/m/n/?] n
    - OPTIONAL:  This allows IPTABLES to match packets based upon their
                 DIFFSERV settings.

  * LENGTH match support (CONFIG_IP_NF_MATCH_LENGTH) [N/m/?] (NEW) n
    - OPTIONAL:  This allows IPTABLES to match packets based upon their
                 packet length.

  * TTL match support (CONFIG_IP_NF_MATCH_TTL) [N/m/?] (NEW) ? n
    - OPTIONAL:  This allows IPTABLES to match packets based upon their
                 TTL settings.

  * tcpmss match support (CONFIG_IP_NF_MATCH_TCPMSS) [N/y/m/?] m
    - OPTIONAL: (Module) Recommended :  This option allows users to examine the MSS
value in
                 TCP SYN packets.  This is an advanced knob but can be very valuable
in
                 troubleshooting MTU problems.

  * Connection state match support (CONFIG_IP_NF_MATCH_STATE) [M/n/?]  m
    - YES: (Module) Recommended : This option allows for Stateful tracking of network
           connections.

  * Unclean match support (EXPERIMENTAL) (CONFIG_IP_NF_MATCH_UNCLEAN) [N/y/m/?] y
```

```
    - YES: (Module) Recommended :  This option allows for connection tracking on odd
packets.
           It cal also help in the detection of possibly malicious packets.
            This can be a valuable tool in tracking hostile people on the network.

  * Owner match support (EXPERIMENTAL) (CONFIG_IP_NF_MATCH_OWNER) [N/y/m/?] n
    - OPTIONAL:  This option allows IPTABLES to match traffic based upon the
                 user login, group, etc. who created the traffic.

  * Packet filtering (CONFIG_IP_NF_FILTER) [N/y/m/?] ? y
    - YES: (Module) This option allows for the kernel to be able filter traffic at
            the INPUT, FORWARDING, and OUTPUT traffic points.

    * REJECT target support (CONFIG_IP_NF_TARGET_REJECT) [N/y/m/?] (NEW) y
      - YES: (Module) With this option, a packet firewall can send an ICMP Reject
packet
            back to the originator when a packet is blocked.

  * MIRROR target support (EXPERIMENTAL) (CONFIG_IP_NF_TARGET_MIRROR) [N/y/m/?]
(NEW) n
    - OPTIONAL: This option allows the packet firewall to mirror the exact same
                network packet back to the originator when it is supposed to be
                blocked.  This is similar to the REJECT option above but it actually
                sends the original packet back to the originator.  i.e. a
                hostile user could actually portscan themselves.


  * Full NAT (CONFIG_IP_NF_NAT) [M/n/?] m
    - YES: (Module) This option enables the future menus to enable Masquerading,
           PORTFWing, Full (1:1) NAT, etc.


  * MASQUERADE target support (CONFIG_IP_NF_TARGET_MASQUERADE) [M/n/?] (NEW) m
    - YES: (Module) This option specifically enables Masquerade into the
           kernel

  * REDIRECT target support (CONFIG_IP_NF_TARGET_REDIRECT) [N/y/m/?] n
    - OPTIONAL: Not needed for normal MASQ functionality though people who
                want to do transparent proxy via Squid will want this.

  * Basic SNMP-ALG support (EXPERIMENTAL) (CONFIG_IP_NF_NAT_SNMP_BASIC) [N/m/?] n
    - OPTIONAL: This enables IPTABLES to properly NAT internal SNMP packets so
                that machines with duplicate addressing ranges can be properly
                managed.


  * Packet mangling (CONFIG_IP_NF_MANGLE) [N/y/m/?] y
    - YES: (Module) This option allows for advanced IPTABLES packet manipulation
           options.


  * TOS target support (CONFIG_IP_NF_TARGET_TOS) [N/y/m/?] (NEW) n
    - OPTIONAL: Enables the kernel to modify the TOS field in a packet
           before routing it on
```

```
  * MARK target support (CONFIG_IP_NF_TARGET_MARK) [N/y/m/?] (NEW) m
    - OPTIONAL: (Module) Recommended : This enables the kernel to manipulate
                packets based upon the MARK field.  This can be used for PORTFW
                as well as many other things.

  * LOG target support (CONFIG_IP_NF_TARGET_LOG) [N/y/m/?]  m
    - YES: (Module)  This allows for the logging of packets before they are accepted,
           denied, rejected, etc.

  * TCPMSS target support (CONFIG_IP_NF_TARGET_TCPMSS) [N/y/m/?] ? m
    - YES: (Module) This option help some people with MTU problems.  Typically,
           most users have to set their Internet connection's MTU to
           1500 as well as ALL internal machines to 1500.  With this
           option, this whole MTU issue might be finally solved.

  * ipchains (2.2-style) support (CONFIG_IP_NF_COMPAT_IPCHAINS) [N/y/m/?] m
    - OPTIONAL: (Module) Recommended : If you have an existing IPCHAINS ruleset
           (2.2.x kernels) and enable this option, you can continue to use the
           IPCHAINS program and the majority of your old ruleset except for the
           use of any 2.2.x kernel-specific modules.  Please note that if this
           IPCHAINS module is loaded, ALL IPTABLES modules will be non-
           operational.  This is an either/or deal only intended for legacy
           rulesets.

  * ipfwadm (2.0-style) support (CONFIG_IP_NF_COMPAT_IPFWADM) [N/y/m/?] n
    - OPTIONAL: If you have an existing IPFWADM ruleset (2.0.x kernels) and
           enable this option, you can continue to use the IPFWADM program and
           the majority of your old ruleset except for the use of any 2.0.x
           kernel-specific modules.   Please note that if this IPFWADM module
           is loaded, ALL IPTABLES modules will be non operational.  This is
           an either/or deal only intended to support legacy
rulesets.


    == Non-MASQ options skipped
    ==   (IPv6, khttpd, ATM, IPX, AppleTalk, etc.) --

  * Fast switching (read help!) (CONFIG_NET_FASTROUTE) [N/y/?] n
    - NO: This performance optimization is NOT compatible with IP MASQ and/or
          packet filtering


    == Non-MASQ options skipped
    == (QoS, Telephony, IDE, SCSI, 1394FW, I2O, etc)

      == Don't forget to compile in support for hardware that you might need:
      ==   IDE:    HDs, CDROMs, etc.
      ==   SCSI:   HDs, CDROMs, etc.


[ Network device support ]

  * Network device support (CONFIG_NETDEVICES) [Y/n/?]
    - YES: Enables the Linux Network device sublayer
```

```
    == Non-MASQ options skipped
    ==   (Arcnet)


  * Dummy net driver support (CONFIG_DUMMY) [M/n/y/?]
    - YES:  Though OPTIONAL, this option can help when debugging problems

    == Non-MASQ options skipped
    == (EQL, etc..)

    == Don't forget to compile in support for hardware that you might need:
    ==   NICs:   eth, tr, etc.
    ==   MODEMs: ppp (ppp async) and/or slip
    ==   WANs:   T1, T3, ISDN, etc.
    ==   ISDN:   for internal ISDN modems


    == Non-MASQ options skipped
    ==   (Amateur Radio, IrDA, ISDN, USB, etc.)


[ Character devices ]

    == Don't forget to compile in serial port support if you are a modem user
    == Don't forget to compile in mouse support

    == Non-MASQ options skipped
    ==   (I2C, Watchdog cards, Ftape, Video for Linux, etc. )


[ File systems ]

    == Non-MASQ options skipped
    ==   (Quota, ISO9660, NTFS, etc )

  * /proc filesystem support (CONFIG_PROC_FS) [Y/n/?]
    - YES:  Required to dynamically configure the Linux forwarding
            and NATing systems


    == Non-MASQ options skipped
    ==   (Console drivers, Sound, USB, Kernel Hacking)
```

So go ahead and select "exit" and you should be prompted to save your config.

NOTE: These are just the kernel components you need for IP Masquerade networking support. You will need to select whatever other options needed for your specific setup. If you want more information on what each one of these kernel modules does, please see the FAQ section of this HOWTO for details.

- Now compile the kernel (make dep; make clean; make bzImage; make modules; make modules_install) , etc. Again, it is beyond the scope of this HOWTO if you have problems compiling your kernel. Please see Section 2.6 for URLs to the KERNEL howto, etc.
- You will then have move over the kernel binary, update your bootloader (LILO, Grub, etc.), and reboot. If you have questions about kernel compiling, I highly recommend to consult some of the URLs mentioned above in this section.

3.2.2. Compiling Linux 2.2.x Kernels

Please see Section 2.7 for any required software, patches, etc.

- First of all, you need the kernel source for 2.2.x (preferably the latest kernel version)

    - NOTE #1: --- UPDATE YOUR KERNEL --- Linux 2.2.x kernels less than version 2.2.20 contain several different security vulnerabilities (some were MASQ specific). Kernels less than 2.2.20 have a few local vulnerabilities. Kernel versions less than 2.2.16 have a TCP root exploit vulnerability and versions less than 2.2.11 have a IPCHAINS fragmentation bug. Because of these issues, users running a firewall with strong IPCHAINS rulesets are open to possible instrusion. Please upgrade your kernel to a fixed version.
    - NOTE #2: As the 2.2.x train progressed, the compile-time options keep on changing. As of this version, this section reflects the settings for a 2.2.20 kernel.

      If you are running either a newer or older kernel version, the dialogs will look different. It is recommended that you update to the newest kernel for added capability and stability of the system.
- If this is your first time compiling the kernel, don't be scared. In fact, it's rather easy and it's covered in several URLs found in Section 2.7. Please note that the instructions included here is just one way to do build a kernel. Please see the Kernel HOWTO for full details.

  NOTE: Please notice that it isn't recommended to put the new kernel sources into /usr/src/linux. You should leave the original kernel sources that came with your Linux distribution in /usr/src/linux. For more details on this topic, please read the "README" file in the top level directory of your kernel sources.
- For this HOWTO example, create a directory called `/usr/src/kernel`. Next, "cd" into this directory and download the newest 2.2.x kernel sources into it. Once downloaded, issue the following command (if the file ends in a .tar.gz): `tar xvzf linux-2.2.x.tar.gz` or (if the file ends in a .tar.bzip2): `tar xyvf linux-2.2.x.tar.bz2`. Please substitute the "x" in the 2.2.x filename with the Linux 2.2 kernel version you downloaded.

  NOTE: Some Linux distributions use the "I" option instead of the "y" option to decompress bzip2 archives.

  Once uncompressed, I recommend that you rename the directory from "linux" to "linux-2.2.x" for clarity. To do this, run the command `mv linux linux-2.2.x`. Next, make sure there is a directory or symbolic link pointing to `/usr/src/kernel/linux` ie. run the command: `ln -s /usr/src/kernel/linux-2.2.x /usr/src/kernel/linux`o again subsituting the "x" for your proper kernel version.
- Apply any appropriate or optional patches to the kernel source code. By default, stock Linux kernels do not require any specific patching in order for the system to work. Features like PPTP/IPSEC masqurading are already built-in in the newest kernels but other tools like Xwindows forwarders are optional. Please refer to Section 2.7 for URLs and the IP Masquerade Resources for up-to-date information and patch URLs.
- Now that the kernel is patched up (if required), here are the MINIMUM kernel configuration options required to enable IP Masquerade functionality. Please understand that this HOWTO illustrates just ONE way to compile a kernel. The main difference from this method vs. a different one is some people wish to compile things either as modules OR monolithically right into the kernel. Basically, compiling things as modules gives you added flexibility to what is or isn't installed into the kernel (reduces unneeded memory use and allow for drop-in upgrades [no need to reboot]) BUT they add more complexity to your configuration. On the flip side, compiling things directly into the kernel makes things simpler BUT you loose a level of flexibility. The following example is a mixture of both built-in AND modules.

  Side Note: It is assumed that you will also configure the kernel to use your other installed hardware such as network interfaces, optional SCSI controllers, etc. as well. Please refer to the Linux Kernel HOWTO and the kernel source's README file and Documentation/ directory for detailed help on compiling a kernel.

Please note the **YES or NO ANSWERS** to the following. Not all options will be available without the proper kernel patches described later in this HOWTO.

Run the following commands to configure your kernel:

- `cd /usr/src/kernel/linux`
- `make menuconfig`

The following kernel prompts reflect a 2.2.20 kernel:

```
[ Code maturity level options ]

  * Prompt for development and/or incomplete code/drivers (CONFIG_EXPERIMENTAL) [Y/
n/?]
    - YES: though not entirely required for IP MASQ, this option allows the kernel
           to create possible additional MASQ modules such as PORTFW, etc.

  == Non-MASQ options skipped
  ==   (CPU, memory, MTRR, SMP, etc.)


[ Loadable module support ]

  * Enable loadable module support (CONFIG_MODULES) [Y/n/?] y
    - YES: allows you to load kernel IP MASQ modules

  * Set version information on all symbols for modules (CONFIG_MODVERSIONS) [N/y/?] y
    - YES: allows newer kernels to load older modules if possible

  * Kernel module loader (CONFIG_KMOD) [Y/n/?] y
    - OPTIONAL: Recommended : allows the kernel to load various kernel modules as
        it needs them


[ General setup ]

  * Networking support (CONFIG_NET) [Y/n/?]
    - YES: This enables the network subsystem

  == Non-MASQ options skipped
  ==   (PCI, kernel binaries, specific hardware options, etc.)


  * Sysctl support (CONFIG_SYSCTL) [Y/n/?]
    - YES:  Enables the ability to enable disable options such as forwarding,
      dynamic IPs, etc. via the /proc interface


[ Block devices ]

  == Non-MASQ options skipped
  ==   (kernel binaries, power management, PnP, IDE, SCSI, etc.)

    == Don't forget to compile in support for hardware that you might need:
```

```
    ==   IDE controllers, HDs, CDROMs, etc.


[ Networking options ]


  * Packet socket (CONFIG_PACKET) [Y/m/n/?] y
    - YES: Though this is OPTIONAL, this recommended feature will allow you
           to use TCPDUMP to debug any problems with IP MASQ

  * Kernel/User netlink socket (CONFIG_NETLINK) [Y/n/?] y
    - OPTIONAL: Recommended :  This feature will allow the logging of
           advanced firewall issues such as routing messages, etc

  * Routing messages (CONFIG_RTNETLINK) [Y/n/?] y
    - OPTIONAL: If you enabled the CONFIG_NETLINK option above, this option
           will send routing messages and other information to SYSLOG.

  * Netlink device emulation (CONFIG_NETLINK_DEV) [N/y/m/?] (NEW) n
    - NO:  This option does not have anything to do with packet firewall
           logging

  * Network firewalls (CONFIG_FIREWALL) [Y/n/?] y
    - YES: Enables the kernel to be comfigured by the IPCHAINS firewall tool

  * Socket Filtering (CONFIG_FILTER) [Y/n/?] y
    - OPTIONAL:  Though this doesn't have anything do with IPMASQ, if you
          plan on implimenting a DHCP server on the internal network, you
          WILL need this option.

  * Unix domain sockets (CONFIG_UNIX) [Y/m/n/?] y
    - YES:  This enables the UNIX TCP/IP sockets mechanisms

  * TCP/IP networking (CONFIG_INET) [Y/n/?] y
    - YES: Enables the TCP/IP protocol

  * IP: multicasting (CONFIG_IP_MULTICAST) [N/y/?] y
    - OPTIONAL:  You can enable this if you want to be able to receive
                 Multicast traffic.  Please note that your ISP must
                 support Multicast as well for this all to work

  * IP: advanced router (CONFIG_IP_ADVANCED_ROUTER) [Y/n/?] n
    - OPTIONAL:  Though there is nothing in this section mandatory for
                 Masquerade, some specific options might be useful

  * IP: kernel level autoconfiguration (CONFIG_IP_PNP) [N/y/?] ?
    - NO:  Not needed for normal MASQ functionality

  * IP: firewalling (CONFIG_IP_FIREWALL) [Y/n/?] y
    - YES: This enables the kernel to support packet filtering, NAT, etc.

  * IP: firewall packet netlink device (CONFIG_IP_FIREWALL_NETLINK) [Y/n/?] n
    - OPTIONAL: Though this is OPTIONAL, this feature will allow IPCHAINS to
                copy some packets to UserSpace tools for additional checks
```

```
  * IP: transparent proxy support (CONFIG_IP_TRANSPARENT_PROXY) [N/y/?] n
    - OPTIONAL:  Not needed for normal MASQ functionality though people who
          want to do transparent proxy via Squid will want this.  Please note
          that there is a PERFORMANCE PENALTY enabling this feature.

  * IP: masquerading (CONFIG_IP_MASQUERADE) [Y/n/?] y
    - YES: Enable IP Masquerade to re-address specific internal to external
          TCP/IP packets

  * IP: ICMP masquerading (CONFIG_IP_MASQUERADE_ICMP) [Y/n/?] y
    - YES: Enable support for masquerading ICMP ping packets (ICMP error
          codes will be MASQed regardless).  This is an important feature
          for troubleshooting connections.

  * IP: masquerading special modules support (CONFIG_IP_MASQUERADE_MOD) [Y/n/?] y
    - YES: Though OPTIONAL, this enables the option to later enable other
          modules like the PORTFW to give external computers a directly
          connection to specified internal MASQed machines.

  * IP: ipautofw masq support (EXPERIMENTAL) (CONFIG_IP_MASQUERADE_IPAUTOFW) [N/y/
m/?] n
    - NO:  NOT recommended : IPautofw is a legacy method of port forwarding.  It
          is mainly old code and has been found to have some issues.

  * IP: ipportfw masq support (EXPERIMENTAL) (CONFIG_IP_MASQUERADE_IPPORTFW) [Y/m/
n/?] y
    - OPTIONAL: Recommended : This enables PORTFW which allows external computers
          on the Internet to directly communicate to specified internal MASQed
          machines.  This feature is typically used to allow access to internal
          SMTP, TELNET, and WWW servers.  Please note that FTP port forwarding
          needs an additional patch, as described in the FAQ section of the MASQ
          HOWTO.  Please see the this FAQ section in the HOWTO for additional
          information.

  * IP: ip fwmark masq-forwarding support (EXPERIMENTAL) (CONFIG_IP_MASQUERADE_MFW)
[Y/m/n/?] y
    - OPTIONAL:  This is a NEW method of performing PORTFW-like functionality which
is
          similar to how the new 2.4.x kernels do things.  With this option,
IPCHAINS
          can mark packets that should have additional work done upon it.  Using a
          UserSpace tool, much like IPMASQADM or IPPORFW, IPCHAINS would then
          do things like re-address the packets, change their TOS value, etc.
          Currently, this code is less tested than PORTFW but it looks promising.
          For now, this HOWTO recommends to use IPMASQADM and IPPORTFW.  If you
          have specific thoughts or comments on MFW, please email dranch.

  * IP: optimize as a router not host (CONFIG_IP_ROUTER) [Y/n/?] y
    - YES:  This optimizes the kernel for the network subsystem, though it
           isn't well known if this makes a siginificant performance difference
           or not.

  == Non-MASQ options skipped
  ==   ( autoconf, tunneling, GRE )
```

```
  * IP: multicast routing (CONFIG_IP_MROUTE) [N/y/?] n
    - OPTIONAL:  Though not needed for IPMASQ, enabling this feature will
                 let you route multicast traffic through your Linux box.
                 Please note that this requires that your ISP be multicast
                 enabled as well.


    == Non-MASQ options skipped
    ==  (Aliasing, ARPd)

  * IP: TCP syncookie support (disabled per default) (CONFIG_SYN_COOKIES) [Y/n/?]
    - YES: Recommended : for basic TCP/IP network security

  * IP: GRE tunnels over IP (CONFIG_NET_IPGRE) [N/y/m/?]
    - NO:   This OPTIONAL selection is to enable PPTP and GRE tunnels through
            the IP MASQ box

    == Non-MASQ options skipped
    ==   (aliasing, ARPd)


  * IP: TCP syncookie support (not enabled per default) (CONFIG_SYN_COOKIES) [Y/n/?]
    - YES: HIGHLY recommended for basic TCP/IP network security

    == Non-MASQ options skipped
    ==  (RARP)


  * IP: Allow large windows (not recommended if <16Mb of memory) *
(CONFIG_SKB_LARGE) [Y/n/?]
    - YES:  This is recommended to optimize Linux's TCP window

    == Non-MASQ options skipped
    ==   (IPv6, IPX, WAN router, etc.)

  * Fast switching (read help!) (CONFIG_NET_FASTROUTE) [N/y/?] n
    - NO: This performance optimization is NOT compatible with IP MASQ and/or
          packet filtering


  == Non-MASQ options skipped
  == (Slow CPU, Telephony, SCSI, I2O, etc. )

    == Don't forget to compile in support for hardware that you might need:
    ==   SCSI:   HDs, CDROMs, etc.


[ Network device support ]

  * Network device support (CONFIG_NETDEVICES) [Y/n/?]
    - YES: Enables the Linux Network device sublayer


  == Non-MASQ options skipped
```

```
  ==   (Arcnet)


  * Dummy net driver support (CONFIG_DUMMY) [M/n/y/?]
    - YES:  Though OPTIONAL, this option can help when debugging problems


  == Non-MASQ options skipped
  == (EQL, NICs, Wireless, IrDA, ISDN, etc..)

    == Don't forget to compile in support for hardware that you might need:
    ==   NICs:   eth, tr, etc.
    ==   MODEMs: ppp and/or slip
    ==   WANs:   T1, T3, ISDN, etc.
    ==   ISDN:   for internal ISDN modems


 [ Character devices ]

  == Don't forget to compile in serial port support for modem users
  == Don't forget to compile in mouse support


  == Non-MASQ options skipped
  ==   (I2C, Watchdog cards, Ftape, Video for Linux, USB, etc. )


[ File systems ]

  == Non-MASQ options skipped
  ==   (Quota, ISO9660, NTFS, etc )


  * /proc filesystem support (CONFIG_PROC_FS) [Y/n/?]
    - YES:  Required to dynamically configure the Linux forwarding
            and NATing systems


  == Non-MASQ options skipped
  ==   (network fs, NLS, video section, sound, kernel hacking)
```

So go ahead and "exit" and you should be prompted to save your config.

NOTE: These are just the components you need for IP Masquerade. You will need to select whatever other options needed for your specific setup.

- Now compile the kernel (make dep; make clean; make bzImage; make modules; make modules_install) , etc. Again, it is beyond the scope of this HOWTO if you have problems compiling your kernel. Please see Section 2.7 for URLs to the KERNEL howto, etc.
- You will then have move over the kernel binary, update your bootloader (LILO, Grub, etc.), and reboot. If you have questions about kernel compiling, I highly recommend to consult some of the URLs above in this section.

3.2.3. Compiling Linux 2.0.x Kernels

Please see Section 2.8 for any required software, patches, etc.

- First of all, you need the kernel source for 2.0.x (preferably the latest kernel version)
    - As the 2.0.x train progress, the compile-time options keep on changing. As of this version, this section reflects the settings for a 2.0.39 kernel.
- If this is your first time compiling the kernel, don't be scared. In fact, it's rather easy and it's covered in several URLs found in Section 2.8. Please note that the instructions included here is just one way to do build a kernel. Please see the Kernel HOWTO for full details.

  NOTE: Please notice that it isn't recommended to put the new kernel sources into /usr/src/linux. You should leave the original kernel sources that came with your Linux distribution in /usr/src/linux. For more details on this topic, please read the "README" file in the top level directory of your kernel sources.
- For this HOWTO example, create a directory called `/usr/src/kernel`. Next, "cd" into this directory and download the newest 2.0.x kernel sources into it. Once downloaded, issue the following command: `tar xvzf linux-2.0.x.tar.gz` . Please substitute the "x" in the 2.0.x filename with the Linux 2.0 kernel version you downloaded.

  Once uncompressed, I recommend that you rename the directory from "linux" to "linux-2.0.x" for clarity. To do this, run the command `mv linux linux-2.0.x`. Next, make sure there is a directory or symbolic link pointing to `/usr/src/kernel/linux` ie. run the command: `ln -s /usr/src/kernel/linux-2.0.x /usr/src/kernel/linux`o again subsituting the "x" for your proper kernel version.
- Apply any appropriate or optional patches to the kernel source code. By default, stock Linux kernels do not require any specific patching in order for the system to work. Features like IPPORTFW, PPTP, and Xwindows forwarders are optional but very useful. Please refer to Section 2.8 for URLs and the IP Masquerade Resources for up-to-date information and patch URLs.
- Now that the kernel is patched up (if required), here are the MINIMUM kernel configuration options required to enable IP Masquerade functionality. Please understand that this HOWTO illustrates just ONE way to compile a kernel. The main difference from this method vs. a different one is some people wish to compile things either as modules OR monolithically right into the kernel. Basically, compiling things as modules gives you added flexibility to what is or isn't installed into the kernel (reduces unneeded memory use and allow for drop-in upgrades [no need to reboot]) BUT they add more complexity to your configuration. On the flip side, compiling things directly into the kernel makes things simpler BUT you loose a level of flexibility. The following example is a mixture of both built-in AND modules.

  Side Note: It is assumed that you will also configure the kernel to use your other installed hardware such as network interfaces, optional SCSI controllers, etc. as well. Please refer to the Linux Kernel HOWTO and the kernel source's "`README`" file and "`Documentation/`" directory for detailed help on compiling a kernel.

Please note the **YES or NO ANSWERS** to the following options. Not all options will be available without the proper kernel patches described later in this HOWTO:

Run the following commands to configure your kernel:

- `cd /usr/src/kernel/linux`
- `make menuconfig`

The following kernel prompts reflect a 2.0.39 kernel:

```
[ Code maturity level options ]

  * Prompt for development and/or incomplete code/drivers (CONFIG_EXPERIMENTAL) [Y/
n/?]
    - YES: this will allow you to later select the IP Masquerade feature code


[ Loadable module support ]

  * Enable loadable module support (CONFIG_MODULES) [Y/n/?] y
    - YES: allows you to load kernel IP MASQ modules

  * Set version information on all module symbols (CONFIG_MODVERSIONS) [N/y/?] y
    - YES: allows newer kernels to load older modules if possible

  * Kernel daemon support (e.g. autoload of modules) (CONFIG_KERNELD) [N/y/?] y
    - OPTIONAL: Recommended : allows the kernel to load various kernel modules as
         it needs them


[ General setup ]

  == Non-MASQ options skipped
  ==   (FPU, memory)

  * Networking support (CONFIG_NET) [Y/n/?] y
    - YES: Enables the network subsystem

  == Non-MASQ options skipped
  ==   (memory, PCI, binary format, APM, etc.)

    == Don't forget to compile in support for hardware that you might need:
    ==   IDE controllers, HDs, CDROMs, etc.


[ Networking options ]

  * Network firewalls (CONFIG_FIREWALL) [Y/n/?] y
    - YES: Enables the IPFWADM firewall tool

  == Non-MASQ options skipped
  ==   (Aliasing)


  * TCP/IP networking (CONFIG_INET) [Y/n/?] y
    - YES: Enables the TCP/IP protocol

  * IP: forwarding/gatewaying (CONFIG_IP_FORWARD) [N/y/?] y
    - YES: Enables Linux network packet forwarding and routing
           - Controlled by IPFWADM

  * IP: multicasting (CONFIG_IP_MULTICAST) [N/y/?] y
    - OPTIONAL:  You can enable this if you want to be able to receive
                 Multicast traffic.  Please note that your ISP must
                 support Multicast as well for this all to work
```

```
  * IP: syn cookies (CONFIG_SYN_COOKIES) [Y/n/?] y
    - YES: HIGHLY recommended for basic network security

  * IP: firewalling (CONFIG_IP_FIREWALL) [Y/n/?] y
    - YES: Enable the packet firewall features

  * IP: firewall packet logging (CONFIG_IP_FIREWALL_VERBOSE) [Y/n/?] y
    - YES: Allows the kernel to report back on various packets traversing
           the firewall.

  * IP: masquerading (CONFIG_IP_MASQUERADE [Y/n/?] y
    - YES: Enable the kernel to perform IP MASQ NAT functionality

  * IP: ipautofw masquerade support (EXPERIMENTAL) (CONFIG_IP_MASQUERADE_IPAUTOFW)
[Y/n/?] n
    - NO:  NOT Recommended : IPautofw is a legacy method of TCP/IP port forwarding.
           Though IPautofw works, IPPORTFW is a better choice.


  * IP: ipportfw masq support (EXPERIMENTAL) (CONFIG_IP_MASQUERADE_IPPORTFW) [Y/n/?]
y
    - YES: This option is ONLY AVAILABLE VIA A PATCH for the 2.0.x kernels.
           With this option, external computers on the Internet can directly
           communicate to specified internal MASQed machines.  This feature is
           typically used to access internal SMTP, TELNET, and WWW servers.
           FTP port forwarding sometimes might require an additional patch as
           described in the FAQ section.  Additional information on port
           forwarding is available in the Forwards section of this HOWTO.


  * IP: MS PPTP masq support (EXPERIMENTAL) (CONFIG_IP_MASQUERADE_PPTP) [N/y/?]
(NEW) n
    - OPTIONAL: Enabling this feature will allow internal MASQ clients to
          properly connect to PPTP servers on the Internet.

  * IP: MS PPTP Call ID masq support (CONFIG_IP_MASQUERADE_PPTP_MULTICLIENT) [N/y/?]
(NEW) n
    - OPTIONAL:  If you enabled the CONFIG_IP_MASQUERADE_PPTP above, this
          option will allow for multiple internal PPTP clients behind the MASQ
          server to communicate to the same PPTP server.

  * IP: MS PPTP masq debugging (DEBUG_IP_MASQUERADE_PPTP) [N/y/?] n
    - OPTIONAL:  NOT recommended : This is not required for IP MASQ or MASQing PPTP
           connections unless you need additional troubleshooting help.  If enabled,
           this can fill up your logs quickly.

  * IP: MS PPTP masq verbose debugging (DEBUG_IP_MASQUERADE_PPTP_VERBOSE) [N/y/?]
(NEW) n
    - OPTIONAL: NOT Recommended : If you enabled the DEBUG_IP_MASQUERADE_PPTP
           option above, this will make the logging even more verbose.

  * IP: IPSEC ESP & ISAKMP masq support (EXPERIMENTAL) *
(CONFIG_IP_MASQUERADE_IPSEC) [N/y/?] m
    - OPTIONAL: This option allows for some forms of IPSEC tunnels to be
```

```
          masquraded

  * IP: IPSEC masq table lifetime (minutes) (CONFIG_IP_MASQUERADE_IPSEC_EXPIRE) *
[30] (NEW)
    - OPTIONAL: This feature allows to change the MASQ table timeouts so that
      idle IPSEC tunnels won't be prematurely disconnected.

  * IP: Disable inbound ESP destination guessing *
(CONFIG_IP_MASQUERADE_IPSEC_NOGUESS) [N/y/?] n
    - OPTIONAL: This feature allows the kernel to guess where the fully encrypted
IPSEC VPN
          might be going and add it to the MASQ table.

  * IP: IPSEC masq debugging (DEBUG_IP_MASQUERADE_IPSEC) [N/y/?] ? n
    - OPTIONAL:  NOT recommended : This is not required for IP MASQ or MASQing IPSEC
          connections unless you need additional troubleshooting help.  If enabled,
          this can fill up your logs quickly.

  * IP: IPSEC masq verbose debugging (DEBUG_IP_MASQUERADE_IPSEC_VERBOSE) [N/y/?]
(NEW) n
    - OPTIONAL: NOT Recommended : If you enabled the DEBUG_IP_MASQUERADE_IPSEC
          option above, this will make the logging even more verbose.


  * IP: ICMP masquerading (CONFIG_IP_MASQUERADE_ICMP) [Y/n/?]
    - YES: Enable support for masquerading ICMP packets. Though thought of as
          optional, many programs will NOT function properly with out ICMP
          support.

  * IP: transparent proxy support (EXPERIMENTAL) (CONFIG_IP_TRANSPARENT_PROXY) [N/
y/?] n
    - OPTIONAL:  Not needed for normal MASQ functionality though people who
          want to do transparent proxy via Squid will want this.  Please note
          that there is a PERFORMANCE PENALTY enabling this feature.

  * IP: loose UDP port managing (EXPERIMENTAL) (CONFIG_IP_MASQ_LOOSE_UDP) [Y/n/?]
    - YES: This option is ONLY AVAILABLE VIA A PATCH for the 2.0.x kernels.
          With this option, internally masqueraded computers can play
          NAT-friendly games over the Internet.  Explicit details are given
          in the FAQ section of this HOWTO.

  * IP: always defragment (CONFIG_IP_ALWAYS_DEFRAG) [Y/n/?]
    - YES:  This feature optimizes IP MASQ connections

  == Non-MASQ options skipped
  ==   (Accounting)


  * IP: optimize as router not host (CONFIG_IP_ROUTER) [Y/n/?]
    - YES:  This optimizes the kernel for the network subsystem

  == Non-MASQ options skipped
  ==   (Tunneling, Mcast routing, RARP, PMTU, etc.)
```

```
  * IP: Drop source routed frames (CONFIG_IP_NOSR) [Y/n/?]
    - YES: HIGHLY recommended for basic network security

  == Non-MASQ options skipped
  ==   (IPX, Bridging, SCSI, etc.)

    == Don't forget to compile in support for hardware that you might need:
    ==   SCSI controllers, HDs, CDROMs, etc.


[ Network device support ]

  * Network device support (CONFIG_NETDEVICES) [Y/n/?]
    - YES: Enables the Linux Network device sublayer


  == Non-MASQ options skipped
  ==   (Dummy, EQL, PPP, SLIP, NICs, Wireless, etc.)

    == Don't forget to compile in support for hardware that you might need:
    ==   NICs:   eth, tr, etc.
    ==   MODEMs: ppp and/or slip
    ==   WANs:   T1, T3, ISDN, etc.
    ==   ISDN:   for internal ISDN modems


[ File systems ]

  == Non-MASQ options skipped
  ==   (Quota, ISO9660, Codepages, NTFS, etc )


  * /proc filesystem support (CONFIG_PROC_FS) [Y/n/?]
    - YES:  Required to dynamically configure the Linux forwarding
            and NATing systems


 [ Character devices ]

  == Non-MASQ options skipped
  ==   (multi-port serial, parallel, mice, Ftape, Sound, etc. )

    == Don't forget to compile in serial port support for modem users
    == Don't forget to compile in mouse support
```

So go ahead and "exit" and you should be prompted to save your config.

NOTE: These are only components for IP Masquerade functionality. You may need to also select additional options to match your specific network and hardware setup.

- Now compile the kernel (make dep; make clean; make bzImage; make modules; make modules_install) , etc. Again, it is beyond the scope of this HOWTO if you have problems compiling your kernel. Please see Section 2.8 for URLs to the

KERNEL howto, etc.

- You will then have move over the kernel binary, update your bootloader (LILO, Grub, etc.), and reboot. If you have questions about kernel compiling, I highly recommend to consult some of the URLs above in this section.

3.3. Assigning Private Network IP Addresses to the Internal LAN

Since all **INTERNAL MASQed** machines should NOT have official Internet assigned addressees, there must be a specific and accepted way to allocate addresses to those machines without conflicting with anyone else's Internet address.

From the original IP Masquerade FAQ:

RFC 1918 is the official document on which IP addresses are to be used in a non-connected or "private" network. There are 3 blocks of numbers set aside specifically for this purpose.

```
Section 3: Private Address Space

The Internet Assigned Numbers Authority (IANA) has reserved the
following three blocks of the IP address space for private networks:

             10.0.0.0        -   10.255.255.255
             172.16.0.0      -   172.31.255.255
             192.168.0.0     -   192.168.255.255

We will refer to the first block as "24-bit block", the second as "20-bit
block", and the third as "16-bit" block".  Note that the first block is
nothing but a single class A network number, while the second block is a set
of 16 continuous class B network numbers, and the third block is a set of
255
continuous class C network numbers.
```

For the record, my preference is to use the 192.168.0.0 network with a 255.255.255.0 Class-C subnet mask and thus this HOWTO reflects this. Any of the above private networks are valid, but just be SURE to use the correct subnet-mask.

So, if you're using a Class-C network, you should number your TCP/IP enabled machines as 192.168.0.1, 192.168.0.2, 192.168.0.3, .., 192.168.0.x

192.168.0.1 is usually set as the internal gateway or Linux MASQ machine which reaches the external network. Please note that 192.168.0.0 and 192.168.0.255 are the Network and Broadcast address respectively (these addresses are RESERVED). Avoid using these addresses on your machines or your network will not function properly.

3.4. Configuring IP Forwarding Policies

At this point, you should have your kernel and other required packages installed. All network IP addresses, gateway, and DNS addresses should be configured on your Linux MASQ server. If you don't know how to configure your Linux network cards, please

consult the HOWTOs listed in either the 2.4.x Section 2.6, the 2.2.x Section 2.7, or the 2.0.x Section 2.8.

Now, the only thing left to do is to configure the IP firewalling tools to both FORWARD and MASQUERADE the appropriate packets to the correct machine.

**** This section ONLY provides the user with the bare minimum firewall ruleset to get IP Masquerading working.**

Once IP MASQ has been successfully tested (as described later in this HOWTO), please refer to the Stronger IPTABLES ruleset for 2.4.x kernels in Section 6.4.1, the Stronger IPCHAINS ruleset for 2.2.x kernels in Section 6.4.2, and the Stronger IPFWADM ruleset for 2.0.x kernels in Section 6.4.3. Please note that these stronger firewall rulesets are more of a template than anything else. For truly secure firewall rulesets, check out the the requirements section of the HOWTO (2.4.x - Section 2.6, 2.2.x - Section 2.7, 2.0.x - Section 2.8).

Instead of manually typing one of these files by hand, I recommend to simply browse the Example directory or download an archive of all of these rc.firewall-* files.

3.4.1. Configuring IP Masquerade on Linux 2.6.x and 2.4.x Kernels

Please note that IPCHAINS is **no longer the primary firewall configuration tool** for the 2.6.x and 2.4.x kernels. The new kernels now use the IPTABLES toolkit though the new 2.4.x kernels CAN still run most old IPCHAINS or IPFWADM rulesets via a compatiblity module. It should also be noted that when running in this compatibility mode, NO IPTABLES modules can be loaded. The reason for this is that none of the 2.2.x IPMASQ modules are compatible with 2.4.x kernels. For a more detailes for these changes, please see the Section 7.40 section.

Ok, as mentioned before, the `/etc/rc.d/rc.local-*` script can be loaded once after every reboot. The mechanism to load the script varies between different Linux distros (please see below for some exampels). The rc.firewall-iptables script will load all required IPMASQ modules as well as enable the final IPMASQ functionality. For advanced setups, this same file would contain very secure firewall rulesets as well.

Anyway, create the file /etc/rc.d/rc.firewall-iptables with the following initial SIMPLE ruleset:

<rc.firewall-iptables START>

```
#!/bin/sh
#
# rc.firewall-iptables
FWVER=0.76
#
#               Initial SIMPLE IP Masquerade test for 2.6 / 2.4 kernels
#               using IPTABLES.
#
#               Once IP Masquerading has been tested, with this simple
#               ruleset, it is highly recommended to use a stronger
#               IPTABLES ruleset either given later in this HOWTO or
#               from another reputable resource.
#
#
#
# Log:
#       0.76 - Added comments on why the default policy is ACCEPT
```

```
#       0.75 - Added more kernel modules to the comments section
#       0.74 - the ruleset now uses modprobe vs. insmod
#       0.73 - REJECT is not a legal policy yet; back to DROP
#       0.72 - Changed the default block behavior to REJECT not DROP
#       0.71 - Added clarification that PPPoE users need to use
#              "ppp0" instead of "eth0" for their external interface
#       0.70 - Added commented option for IRC nat module
#            - Added additional use of environment variables
#            - Added additional formatting
#       0.63 - Added support for the IRC IPTABLES module
#       0.62 - Fixed a typo on the MASQ enable line that used eth0
#              instead of $EXTIF
#       0.61 - Changed the firewall to use variables for the internal
#              and external interfaces.
#       0.60 - 0.50 had a mistake where the ruleset had a rule to DROP
#              all forwarded packets but it didn't have a rule to ACCEPT
#              any packets to be forwarded either
#            - Load the ip_nat_ftp and ip_conntrack_ftp modules by default
#       0.50 - Initial draft
#

echo -e "\n\nLoading simple rc.firewall-iptables version $FWVER..\n"


# The location of the iptables and kernel module programs
#
#   If your Linux distribution came with a copy of iptables,
#   most likely all the programs will be located in /sbin.  If
#   you manually compiled iptables, the default location will
#   be in /usr/local/sbin
#
# ** Please use the "whereis iptables" command to figure out
# ** where your copy is and change the path below to reflect
# ** your setup
#
#IPTABLES=/sbin/iptables
IPTABLES=/usr/local/sbin/iptables
DEPMOD=/sbin/depmod
MODPROBE=/sbin/modprobe


#Setting the EXTERNAL and INTERNAL interfaces for the network
#
#  Each IP Masquerade network needs to have at least one
#  external and one internal network.  The external network
#  is where the natting will occur and the internal network
#  should preferably be addressed with a RFC1918 private address
#  scheme.
#
#  For this example, "eth0" is external and "eth1" is internal"
#
#
#  NOTE:  If this doesnt EXACTLY fit your configuration, you must
#         change the EXTIF or INTIF variables above. For example:
#
```

```
#            If you are a PPPoE or analog modem user:
#
#               EXTIF="ppp0"
#
#
EXTIF="eth0"
INTIF="eth1"
echo "   External Interface:  $EXTIF"
echo "   Internal Interface:  $INTIF"


#=================================================================
#== No editing beyond this line is required for initial MASQ testing ==


echo -en "   loading modules: "

# Need to verify that all modules have all required dependencies
#
echo "  - Verifying that all kernel modules are ok"
$DEPMOD -a

# With the new IPTABLES code, the core MASQ functionality is now either
# modular or compiled into the kernel.  This HOWTO shows ALL IPTABLES
# options as MODULES.  If your kernel is compiled correctly, there is
# NO need to load the kernel modules manually.
#
#  NOTE: The following items are listed ONLY for informational reasons.
#        There is no reason to manual load these modules unless your
#        kernel is either mis-configured or you intentionally disabled
#        the kernel module autoloader.
#

# Upon the commands of starting up IP Masq on the server, the
# following kernel modules will be automatically loaded:
#
# NOTE:  Only load the IP MASQ modules you need.  All current IP MASQ
#        modules are shown below but are commented out from loading.
# ================================================================

echo "-----------------------------------------------------------------"

#Load the main body of the IPTABLES module - "iptable"
#  - Loaded automatically when the "iptables" command is invoked
#
#  - Loaded manually to clean up kernel auto-loading timing issues
#
echo -en "ip_tables, "
$MODPROBE ip_tables


#Load the IPTABLES filtering module - "iptable_filter"
#  - Loaded automatically when filter policies are activated
```

```
#Load the stateful connection tracking framework - "ip_conntrack"
#
# The conntrack  module in itself does nothing without other specific
# conntrack modules being loaded afterwards such as the "ip_conntrack_ftp"
# module
#
#  - This module is loaded automatically when MASQ functionality is
#    enabled
#
#  - Loaded manually to clean up kernel auto-loading timing issues
#
echo -en "ip_conntrack, "
$MODPROBE ip_conntrack


#Load the FTP tracking mechanism for full FTP tracking
#
# Enabled by default -- insert a "#" on the next line to deactivate
#
echo -en "ip_conntrack_ftp, "
$MODPROBE ip_conntrack_ftp


#Load the IRC tracking mechanism for full IRC tracking
#
# Enabled by default -- insert a "#" on the next line to deactivate
#
echo -en "ip_conntrack_irc, "
$MODPROBE ip_conntrack_irc


#Load the general IPTABLES NAT code - "iptable_nat"
#  - Loaded automatically when MASQ functionality is turned on
#
#  - Loaded manually to clean up kernel auto-loading timing issues
#
echo -en "iptable_nat, "
$MODPROBE iptable_nat


#Loads the FTP NAT functionality into the core IPTABLES code
# Required to support non-PASV FTP.
#
# Enabled by default -- insert a "#" on the next line to deactivate
#
echo -en "ip_nat_ftp, "
$MODPROBE ip_nat_ftp


#Loads the IRC NAT functionality into the core IPTABLES code
# Required to support NAT of IRC DCC requests
#
# Disabled by default -- remove the "#" on the next line to activate
#
#echo -e "ip_nat_irc"
```

```
#$MODPROBE ip_nat_irc

echo "------------------------------------------------------------------------"

# Just to be complete, here is a partial list of some of the other
# IPTABLES kernel modules and their function.  Please note that most
# of these modules (the ipt ones) are automatically loaded by the
# master kernel module for proper operation and don't need to be
# manually loaded.
# ------------------------------------------------------------------------
#
#    ip_nat_snmp_basic - this module allows for proper NATing of some
#                        SNMP traffic
#
#    iptable_mangle    - this target allows for packets to be
#                        manipulated for things like the TCPMSS
#                        option, etc.
#
# --
#
#    ipt_mark       - this target marks a given packet for future action.
#                     This automatically loads the ipt_MARK module
#
#    ipt_tcpmss     - this target allows to manipulate the TCP MSS
#                     option for braindead remote firewalls.
#                     This automatically loads the ipt_TCPMSS module
#
#    ipt_limit      - this target allows for packets to be limited to
#                     to many hits per sec/min/hr
#
#    ipt_multiport  - this match allows for targets within a range
#                     of port numbers vs. listing each port individually
#
#    ipt_state      - this match allows to catch packets with various
#                     IP and TCP flags set/unset
#
#    ipt_unclean    - this match allows to catch packets that have invalid
#                     IP/TCP flags set
#
#    iptable_filter - this module allows for packets to be DROPped,
#                     REJECTed, or LOGged.  This module automatically
#                     loads the following modules:
#
#                     ipt_LOG - this target allows for packets to be
#                               logged
#
#                     ipt_REJECT - this target DROPs the packet and returns
#                                  a configurable ICMP packet back to the
#                                  sender.
#

echo -e "   Done loading modules.\n"
```

```
#CRITICAL:  Enable IP forwarding since it is disabled by default since
#
#           Redhat Users:  you may try changing the options in
#                          /etc/sysconfig/network from:
#
#                       FORWARD_IPV4=false
#                             to
#                       FORWARD_IPV4=true
#
echo "   Enabling forwarding.."
echo "1" > /proc/sys/net/ipv4/ip_forward


# Dynamic IP users:
#
#   If you get your IP address dynamically from SLIP, PPP, or DHCP,
#   enable this following option.  This enables dynamic-address hacking
#   which makes the life with Diald and similar programs much easier.
#
echo "   Enabling DynamicAddr.."
echo "1" > /proc/sys/net/ipv4/ip_dynaddr


# Enable simple IP forwarding and Masquerading
#
#  NOTE:  In IPTABLES speak, IP Masquerading is a form of SourceNAT or SNAT.
#
#  NOTE #2:  The following is an example for an internal LAN address in the
#            192.168.0.x network with a 255.255.255.0 or a "24" bit subnet mask
#            connecting to the Internet on external interface "eth0".  This
#            example will MASQ internal traffic out to the Internet but not
#            allow non-initiated traffic into your internal network.
#
#
#         ** Please change the above network numbers, subnet mask, and your
#         *** Internet connection interface name to match your setup
#


#Clearing any previous configuration
#
#  Unless specified, the defaults for INPUT and OUTPUT is ACCEPT
#    The default for FORWARD is DROP (REJECT is not a valid policy)
#
#   Isn't ACCEPT insecure?  To some degree, YES, but this is our testing
#   phase.  Once we know that IPMASQ is working well, I recommend you run
#   the rc.firewall-*-stronger rulesets which set the defaults to DROP but
#   also include the critical additional rulesets to still let you connect to
#   the IPMASQ server, etc.
#
echo "   Clearing any existing rules and setting default policy.."
$IPTABLES -P INPUT ACCEPT
$IPTABLES -F INPUT
$IPTABLES -P OUTPUT ACCEPT
$IPTABLES -F OUTPUT
```

```
$IPTABLES -P FORWARD DROP
$IPTABLES -F FORWARD
$IPTABLES -t nat -F

echo "   FWD: Allow all connections OUT and only existing and related ones IN"
$IPTABLES -A FORWARD -i $EXTIF -o $INTIF -m state --state ESTABLISHED,RELATED -j
ACCEPT
$IPTABLES -A FORWARD -i $INTIF -o $EXTIF -j ACCEPT
$IPTABLES -A FORWARD -j LOG

echo "   Enabling SNAT (MASQUERADE) functionality on $EXTIF"
$IPTABLES -t nat -A POSTROUTING -o $EXTIF -j MASQUERADE

echo -e "\nrc.firewall-iptables v$FWVER done.\n"
```

<rc.firewall-iptables STOP>

Once you are finished with editing this /etc/rc.d/rc.firewall-iptables ruleset, make it executable by typing in `chmod 700 /etc/rc.d/rc.firewall-iptables`

Now that the firewall ruleset is ready, you need to let it run after every reboot. You could either do this by running it by hand everytime (such a pain) or add it to the boot scripts. We have covered two methods below: Redhat (SyS-V style) and Slackware (BSD style)

1. Redhat and Redhat-derived distros:

- There are two ways to automatically load things in Redhat: /etc/rc.d/rc.local or a init script in /etc/rc.d/init.d/. The first method is the easiest but isn't doing things the SYS-V way. All you have to do is add the line:

  ```
  echo "Loading the rc.firewall-iptables ruleset.. "
  /etc/rc.d/rc.firewall-iptables
  ```

  to the end of the /etc/rc.d/rc.local file and thats it (as described earlier in the HOWTO).

  The problem with this approach is that the firewall isn't executed until the last stages of booting.
- The preferred approach is to have the firewall loaded just after the networking subsystem is loaded. To do this, copy the following file into the /etc/rc.d/init.d directory:

  <firewall-iptables START>

  ```
  #!/bin/sh
  #
  # chkconfig: 2345 11 89
  #
  # description: Loads the rc.firewall-iptables ruleset.
  #
  # processname: firewall-iptables
  # pidfile: /var/run/firewall.pid
  # config: /etc/rc.d/rc.firewall-iptables
  # probe: true

  # ------------------------------------------------------------------------
  ```

```
# v05/24/03
#
# Part of the copyrighted and trademarked TrinityOS document.
# http://www.ecst.csuchico.edu/~dranch
#
# Written and Maintained by David A. Ranch
# dranch@trinnet.net
#
# Updates
# -------
# 05/24/03 - removed a old networking up check that had some
#            improper SGML ampersand conversions.
# ----------------------------------------------------------------------------


# Source function library.
. /etc/rc.d/init.d/functions

# Check that networking is up.

[ "XXXX${NETWORKING}" = "XXXXno" ] && exit 0

[ -x /sbin/ifconfig ] || exit 0

# The location of various iptables and other shell programs
#
#   If your Linux distribution came with a copy of iptables, most
#   likely it is located in /sbin.  If you manually compiled
#   iptables, the default location is in /usr/local/sbin
#
# ** Please use the "whereis iptables" command to figure out
# ** where your copy is and change the path below to reflect
# ** your setup
#
IPTABLES=/usr/local/sbin/iptables


# See how we were called.
case "$1" in
  start)
    /etc/rc.d/rc.firewall-iptables
    ;;

  stop)
    echo -e "\nFlushing firewall and setting default policies to DROP\n"
    $IPTABLES -P INPUT DROP
    $IPTABLES -F INPUT
    $IPTABLES -P OUTPUT DROP
    $IPTABLES -F OUTPUT
    $IPTABLES -P FORWARD DROP
    $IPTABLES -F FORWARD
    $IPTABLES -F -t nat

    # Delete all User-specified chains
    $IPTABLES -X
```

```
    #
    # Reset all IPTABLES counters
    $IPTABLES -Z
    ;;

  restart)
        $0 stop
        $0 start
        ;;

  status)
        $IPTABLES -L
        ;;

  mlist)
    cat /proc/net/ip_conntrack
    ;;

  *)
        echo "Usage: firewall-iptables {start|stop|status|mlist}"
        exit 1
esac

exit 0
```

<firewall-iptables STOP>

With this script in place, all you need to do now is make it executable and then make it load upon reboot. First, make it executable by running:

```
#Redhat-style
#
chmod 700 /etc/rc.d/init.d/firewall-iptables
```

Now, enable the ruleset load upon reboot:

```
#Redhat style
#
/sbin/chkconfig --level=345 firewall-iptables on
```

That's it! Now upon reboot, the firewall will be loaded automatically. Just to make sure, run the following command to see that the firewall should start upon reboot by running the command:

```
#Redhat style
#
chkconfig --list firewall-iptables

#The output should look like:
#
firewall-iptables 0:off 1:off 2:off 3:on 4:on 5:on 6:off
```

2. Slackware:

- There are two ways to load things in Slackware: /etc/rc.d/rc.local or editing the /etc/rc.d/rc.inet2 file. The first method is the

easiest but isn't the most secure (see below). All you have to do is append the following lines to the /etc/rc.d/rc.local file:

```
echo "Loading the rc.firewall-iptables ruleset.."

/etc/rc.d/rc.firewall-iptables
```

The problem with this approach is that if you are running a STRONG firewall ruleset, the firewall isn't executed until the last stages of booting. The preferred approach is to have the firewall loaded just after the networking subsystem is loaded. For now, the HOWTO only covers how to do so using /etc/rc.d/rc.local but if you know what you're doing (it's easy), go ahead and and modify the inet2 startup script to load the /etc/rc.d/rc.firewall-iptables file just after the network is up. If you want a more detailed guide and/or a stronger firewall ruleset, I recommend you check out Section 10 of TrinityOS found in the links section at the bottom of this HOWTO.

Notes on how users might want to change the above firewall ruleset:

You could also have IP Masquerading enabled on a PER MACHINE basis instead of the above method, which is enabling an ENTIRE TCP/IP network. For example, say if I wanted only the 192.168.0.2 and 192.168.0.8 hosts to have access to the Internet and NOT any of the other internal machines. I would change the in the "Enable simple IP forwarding and Masquerading" section (shown above) of the /etc/rc.d/rc.firewall-iptables ruleset.

```
#!/bin/sh
#
# Partial IPTABLES config to enable simple IP forwarding and Masquerading
# v0.61
#
#  NOTE:  The following is an example to allow only IP Masquerading for the
#         192.168.0.2 and 192.168.0.8 machines with a 255.255.255.0 or a
#         "/24" subnet mask connecting to the Internet on interface eth0.
#
#         ** Please change the network number, subnet mask, and the Internet
#         ** connection interface name to match your internal LAN setup
#
echo "  - Setting the default FORWARD policy to DROP"
$IPTABLES -P FORWARD DROP

echo "  - Enabling SNAT (IPMASQ) functionality on $EXTIF"
$IPTABLES -t nat -A POSTROUTING -o $EXTIF -s 192.168.0.2/32 -j MASQUERADE
$IPTABLES -t nat -A POSTROUTING -o $EXTIF -s 192.168.0.8/32 -j MASQUERADE

echo "  - Setting the FORWARD policy to 'DROP' all incoming / unrelated traffic"
$IPTABLES -A INPUT -i $EXTIF -m state --state NEW,INVALID -j DROP
$IPTABLES -A FORWARD -i $EXTIF -m state --state NEW,INVALID -j DROP
```

Common mistakes:

It appears that a common mistake with new IP Masq users is to make the first command simply the following:

```
IPTABLES:
---------
iptables -t nat -A POSTROUTING -j MASQUERADE
```

Do **NOT** make your default policy MASQUERADING. Otherwise, someone can manipulate their routing tables to tunnel straight back through your gateway, using it to masquerade their OWN identity!

Again, you can add these lines to the `/etc/rc.d/rc.firewall-iptables` file, one of the other rc files you prefer, or do it manually every time you need IP Masquerade.

Please see Section 6.4.1 for a detailed guide on a strong IPTABLES ruleset example. For additional details on IPTABLES usage, please refer to http://www.netfilter.org/ for the primary IPTABLES site.

3.4.2. Configuring IP Masquerade on Linux 2.2.x Kernels

Please note that **IPFWADM is no longer the firewall tool** for manipulating IP Masquerading rules for both the 2.1.x and 2.2.x kernels. These new kernels now use the IPCHAINS toolkit. For a more detailed reason for this change, please see Chapter 7.

Create the file /etc/rc.d/rc.firewall-ipchains with the following initial SIMPLE ruleset:

<rc.firewall-ipchains START>

```
#!/bin/sh
#
# rc.firewall-ipchains
#
#     - Initial SIMPLE IP Masquerade test for 2.1.x and 2.2.x kernels
#       using IPCHAINS.
#
#       Once IP Masquerading has been tested, with this simple
#       ruleset, it is highly recommended to use a stronger
#       IPTABLES ruleset either given later in this HOWTO or
#       from another reputable resource.

FWVER="1.23"
#
# 1.23 - Added comments on why the default policy is ACCEPT
# 1.22 - ruleset now uses modprobe instead of insmod
# 1.21 - Added clarification that PPPoE users need to use
#        "ppp0" instead of "eth0" for their external interface
# 1.20 - Updated the script to use environment vars
# 1.01 - Original version


echo -e "\n\nLoading simple rc.firewall-ipchains : version $FWVER..\n"


# The location of the ipchains and kernel module programs
#
```

```
#   If your Linux distribution came with a copy of ipchains,
#   most likely all the programs will be located in /sbin.  If
#   you manually compiled ipchains, the default location will
#   be in /usr/local/sbin
#
# ** Please use the "whereis ipchains" command to figure out
# ** where your copy is and change the path below to reflect
# ** your setup
#
IPCHAINS=/sbin/ipchains
#IPTABLES=/usr/local/sbin/ipchains
DEPMOD=/sbin/depmod
MODPROBE=/sbin/modprobe


#Setting the EXTERNAL and INTERNAL interfaces for the network
#
#  Each IP Masquerade network needs to have at least one
#  external and one internal network.  The external network
#  is where the NATing will occur and the internal network
#  should preferably be addressed with a RFC1918 private addressing
#  scheme.
#
#  For this example, "eth0" is external and "eth1" is internal"
#
#  NOTE:  If this doesnt EXACTLY fit your configuration, you must
#         change the EXTIF or INTIF variables above. For example:
#
#            If you are a PPPoE or analog modem user:
#
#               EXTIF="ppp0"
#
#  ** Please change this to reflect your specific configuration **
#
EXTIF="eth0"
INTIF="eth1"
echo "   External Interface:  $EXTIF"
echo "   Internal Interface:  $INTIF"


# Network Address of the Internal Network
#
#   This example rc.firewall-ipchains file uses the 192.168.0.0 network
#   with a /24 or 255.255.255.0 netmask.
#
#    ** Change this variable to reflect your specific setup **
#
INTLAN="192.168.0.0/24"
echo -e "   Internal Interface:  $INTLAN\n"



# Load all required IP MASQ modules
#
#   NOTE:  Only load the IP MASQ modules you need.  All current IP MASQ modules
```

```
#          are shown below but are commented out from loading.
echo "   loading required IPMASQ kernel modules.."

# Needed to initially load modules
#
$DEPMOD -a

echo -en "   Loading modules: "

# Supports the proper masquerading of FTP file transfers using the PORT method
#
echo -en "FTP, "
$MODPROBE ip_masq_ftp

# Supports the masquerading of RealAudio over UDP.  Without this module,
#       RealAudio WILL function but in TCP mode.  This can cause a reduction
#       in sound quality
#
#echo -en "RealAudio, "
$MODPROBE ip_masq_raudio

# Supports the masquerading of IRC DCC file transfers
#
#echo -en "Irc, "
#$MODPROBE ip_masq_irc


# Supports the masquerading of Quake and QuakeWorld by default.  This modules is
#   for for multiple users behind the Linux MASQ server.  If you are going to
#   play Quake I, II, and III, use the second example.
#
#   NOTE:  If you get ERRORs loading the QUAKE module, you are running an old
#   -----  kernel that has bugs in it.  Please upgrade to the newest kernel.
#
#echo -en "Quake, "
#Quake I / QuakeWorld (ports 26000 and 27000)
#$MODPROBE ip_masq_quake
#
#Quake I/II/III / QuakeWorld (ports 26000, 27000, 27910, 27960)
#$MODPROBE ip_masq_quake 26000,27000,27910,27960


# Supports the masquerading of the CuSeeme video conferencing software
#
#echo -en "CuSeeme, "
#$MODPROBE ip_masq_cuseeme

#Supports the masquerading of the VDO-live video conferencing software
#
#echo -en "VdoLive "
#$MODPROBE ip_masq_vdolive

echo ".  Done loading modules."
```

```
#CRITICAL:  Enable IP forwarding since it is disabled by default since
#
#           Redhat Users:  you may try changing the options in
#                          /etc/sysconfig/network from:
#
#                       FORWARD_IPV4=false
#                             to
#                       FORWARD_IPV4=true
#
echo "   enabling forwarding.."
echo "1" > /proc/sys/net/ipv4/ip_forward


#CRITICAL:  Enable automatic IP defragmenting since it is disabled by default
#           in 2.2.x kernels.  This used to be a compile-time option but the
#           behavior was changed in 2.2.12
#
echo "   enabling AlwaysDefrag.."
echo "1" > /proc/sys/net/ipv4/ip_always_defrag


# Dynamic IP users:
#
#   If you get your IP address dynamically from SLIP, PPP, or DHCP, enable this
#   following option.  This enables dynamic-ip address hacking in IP MASQ,
#   making the life with Diald and similar programs much easier.
#
#echo "   enabling DynamicAddr.."
#echo "1" > /proc/sys/net/ipv4/ip_dynaddr


# Enable the LooseUDP patch which some Internet-based games require
#
#  If you are trying to get an Internet game to work through your IP MASQ box,
#  and you have set it up to the best of your ability without it working, try
#  enabling this option (delete the "#" character).  This option is disabled
#  by default due to possible internal machine UDP port scanning
#  vulnerabilities.
#
#echo "   enabling LooseUDP.."
#echo "1" > /proc/sys/net/ipv4/ip_masq_udp_dloose


#Clearing any previous configuration
#
#  Unless specified, the defaults for INPUT and OUTPUT is ACCEPT
#    The default for FORWARD is REJECT
#
#   Isn't ACCEPT insecure?  To some degree, YES, but this is our testing
#   phase.  Once we know that IPMASQ is working well, I recommend you run
#   the rc.firewall-*-stronger rulesets which set the defaults to DROP but
#   also include the critical additional rulesets to still let you connect to
#   the IPMASQ server, etc.
#
echo "   clearing any existing rules and setting default policy.."
```

```
$IPCHAINS -P input ACCEPT
$IPCHAINS -P output ACCEPT
$IPCHAINS -P forward REJECT
$IPCHAINS -F input
$IPCHAINS -F output
$IPCHAINS -F forward


# MASQ timeouts
#
#   2 hrs timeout for TCP session timeouts
#  10 sec timeout for traffic after the TCP/IP "FIN" packet is received
#  160 sec timeout for UDP traffic (Important for MASQ'ed ICQ users)
#
echo "   setting default timers.."
$IPCHAINS -M -S 7200 10 160


# DHCP:  For people who receive their external IP address from either DHCP or
#        BOOTP for connctions such as DSL or Cablemodem users, it is necessary
#        to use the following before the deny command.
#
#        This example is currently commented out.
#
#
#$IPCHAINS -A input -j ACCEPT -i $EXTIF -s 0/0 67 -d 0/0 68 -p udp

# Enable simple IP forwarding and Masquerading
#
#  NOTE:  The following is an example for an internal LAN address in the
#         192.168.0.x network with a 255.255.255.0 or a "24" bit subnet mask
#         connecting to the Internet on interface eth0.
#
#         ** Please change this network number, subnet mask, and your Internet
#         ** connection interface name to match your internal LAN setup
#
echo "   enabling IPMASQ functionality on $EXTIF"
$IPCHAINS -P forward DENY
$IPCHAINS -A forward -i $EXTIF -s $INTLAN -j MASQ

echo -e "\nrc.firewall-ipchains v$FWVER done.\n"
```

<rc.firewall-ipchains STOP>

Once you are finished with editing the /etc/rc.d/rc.firewall-ipchains ruleset, make it executable by typing in `chmod 700 /etc/rc.d/rc.firewall-ipchains`

Now that the firewall ruleset is ready, you need to let it run after every reboot. You could either do this by running it by hand everytime (such a pain) or add it to the boot scripts. We have covered two methods below: Redhat (SyS-V style) and Slackware (BSD style)

1. Redhat and Redhat-derived distros:

- There are two ways to automatically load things in Redhat: /etc/rc.d/rc.local or a init script in /etc/rc.d/init.d/. The first method

is the easiest but isn't doing things the Sys-V way. All you have to do is add the line:

```
echo "Loading the rc.firewall ruleset.."
/etc/rc.d/rc.firewall-ipchains
```

to the end of the /etc/rc.d/rc.local file and thats it (as described earlier in the HOWTO).

The problem with this approach is that the firewall isn't executed until the last stages of booting. The preferred approach is to have the firewall loaded just after the networking subsystem is loaded. To do this, copy the following file into the /etc/rc.d/init.d directory:

<firewall-ipchains START>

```
#!/bin/sh
#
# chkconfig: 2345 11 89
#
# description: Loads the rc.firewall-ipchains ruleset.
#
# processname: firewall-ipchains
# pidfile: /var/run/firewall.pid
# config: /etc/rc.d/rc.firewall-ipchains
# probe: true

# ------------------------------------------------------------------------
# v08/29/02
#
# Part of the copyrighted and trademarked TrinityOS document.
# http://www.ecst.csuchico.edu/~dranch
#
# Written and Maintained by David A. Ranch
# dranch@trinnet.net
#
# Updates
# -------
#
# ------------------------------------------------------------------------


# Source function library.
. /etc/rc.d/init.d/functions

# Check that networking is up.

# This line no longer work with bash2
#[ ${NETWORKING} = "no" ] && exit 0
# This should be OK.
[ "XXXX${NETWORKING}" = "XXXXno" ] && exit 0

[ -x /sbin/ifconfig ] || exit 0

# The location of various iptables and other shell programs
#
```

```
#   If your Linux distribution came with a copy of iptables, most
#   likely it is located in /sbin.  If you manually compiled
#   iptables, the default location is in /usr/local/sbin
#
# ** Please use the "whereis iptables" command to figure out
# ** where your copy is and change the path below to reflect
# ** your setup
#
IPCHAINS=/sbin/ipchains


# See how we were called.
case "$1" in
  start)
    /etc/rc.d/rc.firewall-ipchains
    ;;

  stop)
    echo -e "\nFlushing firewall and setting default policies to REJECT\n"

    $IPCHAINS -P input REJECT
    $IPCHAINS -P output REJECT
    $IPCHAINS -P forward REJECT

    $IPCHAINS -F input
    $IPCHAINS -F output
    $IPCHAINS -F forward
    ;;

  restart)
    $0 stop
    $0 start
    ;;

  status)
    $IPCHAINS -L
    ;;

  mlist)
    $IPCHAINS -M -L
    ;;

  *)
    echo "Usage: firewall-ipchains {start|stop|status|mlist}"
    exit 1
esac

exit 0
```

<firewall-ipchains STOP>

With this script in place, all you need to do now is make it executable and then make it load upon reboot. First, make it executable by running:

```
#Redhat-style
#
chmod 700 /etc/rc.d/init.d/firewall-ipchains
```

Now, make the ruleset load upon reboot:

```
#Redhat style
#
chkconfig --level=345 firewall-ipchains on
```

That's it! Now upon boot, the firewall will be loaded automatically. Just to make sure, run the command to see that the firewall should start upon reboot by running the command:

```
#Redhat style
#
chkconfig --list firewall-ipchains

#The output should look like:
#
firewall-ipchains 0:off 1:off 2:off 3:on 4:on 5:on 6:off
```

2. Slackware:

- There are two ways to load things in Slackware: /etc/rc.d/rc.local or editing the /etc/rc.d/rc.inet2 file. The first method is the easiest but isn't the most secure (see below). All you have to do is append the following lines to the /etc/rc.d/rc.local file:

```
echo "Loading the rc.firewall-ipchains ruleset.."
/etc/rc.d/rc.firewall-ipchains
```

The problem with this approach is that if you are running a STRONG firewall ruleset, the firewall isn't executed until the last stages of booting. The preferred approach is to have the firewall loaded just after the networking subsystem is loaded. For now, the HOWTO only covers how to do so using /etc/rc.d/rc.local but if you know what you're doing (it's easy), go ahead and and modify the inet2 startup script to load the /etc/rc.d/rc.firewall-ipchains file just after the network is up. If you want a more detailed guide and/or a stronger firewall ruleset, I recommend you check out Section 10 of TrinityOS found in the links section at the bottom of this HOWTO.

Notes on how users might want to change the above firewall ruleset:

You could also have IP Masquerading enabled on a PER MACHINE basis instead of the above method, which is enabling an ENTIRE TCP/IP network. For example, say if I wanted only the 192.168.0.2 and 192.168.0.8 hosts to have access to the Internet and NOT any of the other internal machines. I would change the in the "Enable simple IP forwarding and Masquerading" section (shown above) of the /etc/rc.d/rc.firewall-ipchains ruleset.

```
#!/bin/sh
#
# Enable simple IP forwarding and Masquerading
# v1.01
#
#  NOTE:  The following is an example used in addition to the simple
#         IPCHAINS ruleset anove to allow only IP Masquerading for the
#         192.168.0.2 and 192.168.0.8 machines with a 255.255.255.0 or a
#         "24" bit subnet mask connecting to the Internet on interface $EXTIF.
#
#         ** Please change the network number, subnet mask, and the Internet
#         ** connection interface name to match your internal LAN setup
#
$IPCHAINS -P forward DENY
$IPCHAINS -A forward -i $EXTIF -s 192.168.0.2/32 -j MASQ
$IPCHAINS -A forward -i $EXTIF -s 192.168.0.8/32 -j MASQ
```

Common mistakes:

What appears to be a common mistake with new IP MASQ users is to make the first command:

```
$IPCHAINS -P forward masquerade
```

Do **NOT** make your default policy MASQUERADING. Otherwise, someone can manipulate their routing tables to tunnel straight back through your gateway, using it to masquerade their OWN identity!

Again, you can add these lines to the `/etc/rc.d/rc.firewall-ipchains` file, one of the other rc files you prefer, or do it manually every time you need IP Masquerade.

Please see Section 6.4.2 for a detailed guide on IPCHAINS and a strong IPCHAINS ruleset example. For additional details on IPCHAINS usage, please refer to http://www.netfilter.org/ipchains/ for the primary IPCHAINS site or the Linux IP CHAINS HOWTO Backup site

---

3.4.3. Configuring IP Masquerade on Linux 2.0.x Kernels

Create the file /etc/rc.d/rc.firewall-ipfwadm with the following initial SIMPLE ruleset: <rc.firewall-ipfwadm START>

```
#!/bin/sh
#
# rc.firewall-ipfwadm
#
#  A Initial SIMPLE IP Masquerade setup for 2.0.x kernels using IPFWADM
#
FWVER="2.03"
#
# 2.03 - Added comments on why the default policy is ACCEPT
# 2.02 - Added clarification that PPPoE users need to use
#        "ppp0" instead of "eth0" for their external interface
#
#
#        Once IP Masquerading has been tested, with this simple
#        ruleset, it is highly recommended to use a stronger
#        IPTABLES ruleset either given later in this HOWTO or
#        from another reputable resource.
#
echo -e "\n\nLoading simple rc.firewall-ipfwadm version $FWVER..\n"


#Setting the EXTERNAL and INTERNAL interfaces for the network
#
#  Each IP Masquerade network needs to have at least one
#  external and one internal network.  The external network
#  is where the NATing will occur and the internal network
#  should preferably be addressed with a RFC1918 private addressing
#  scheme.
#
#  For this example, "eth0" is external and "eth1" is internal"
#
#  NOTE:  If this doesnt EXACTLY fit your configuration, you must
#         change the EXTIF or INTIF variables above. For example:
#
#            If you are a PPPoE or analog modem user:
#
#               EXTIF="ppp0"
#
#  ** Please change this to reflect your specific configuration **
#
EXTIF="eth0"
INTIF="eth1"
echo "   External Interface:  $EXTIF"
echo "   Internal Interface:  $INTIF"


# Network Address of the Internal Network
#
#   This example rc.firewall-ipfwadm file uses the 192.168.0.0 network
#   with a /24 or 255.255.255.0 netmask.
#
#    ** Change this variable to reflect your specific setup **
#
INTLAN="192.168.0.0/24"
echo -e "   Internal Interface:  $INTLAN\n"
```

```
# Load all required IP MASQ modules
#
#   NOTE:  Only load the IP MASQ modules you need.  All current available IP
#          MASQ modules are shown below but are commented out from loading.
echo -en "Loading modules: "


# Needed to initially load modules
#
/sbin/depmod -a

# Supports the proper masquerading of FTP file transfers using the PORT method
#
echo -en "FTP, "
/sbin/modprobe ip_masq_ftp

# Supports the masquerading of RealAudio over UDP.  Without this module,
#       RealAudio WILL function but in TCP mode.  This can cause a reduction
#       in sound quality
#
#echo -en "RealAudio, "
#/sbin/modprobe ip_masq_raudio

# Supports the masquerading of IRC DCC file transfers
#
#echo -en "Irc, "
#/sbin/modprobe ip_masq_irc

# Supports the masquerading of Quake and QuakeWorld by default.  These modules
#   are for multiple users behind the Linux MASQ server.  If you are going to
#   play Quake I, II, and III, use the second example.
#
#   NOTE:  If you get ERRORs loading the QUAKE module, you are running an old
#   -----  kernel that has bugs in it.  Please upgrade to the newest kernel.
#
#echo -en "Quake, "
#Quake I / QuakeWorld (ports 26000 and 27000)
#/sbin/modprobe ip_masq_quake
#
#Quake I/II/III / QuakeWorld (ports 26000, 27000, 27910, 27960)
#/sbin/modprobe ip_masq_quake 26000,27000,27910,27960

# Supports the masquerading of the CuSeeme video conferencing software
#
#echo -en "CuSeeme, "
#/sbin/modprobe ip_masq_cuseeme

#Supports the masquerading of the VDO-live video conferencing software
#
#echo -en "VdoLive, "
#/sbin/modprobe ip_masq_vdolive

echo ".  Done loading modules."
```

```
#CRITICAL:  Enable IP forwarding since it is disabled by default
#
#           Redhat Users:  you may try changing the options in
#                          /etc/sysconfig/network from:
#
#                       FORWARD_IPV4=false
#                             to
#                       FORWARD_IPV4=true
#
echo "   enabling forwarding.."
echo "1" > /proc/sys/net/ipv4/ip_forward


#CRITICAL:  Enable automatic IP defragmenting since it is disabled by default
#
#           This used to be a compile-time option but the behavior was changed
#           in 2.2.12.  This option is required for both 2.0 and 2.2 kernels.
#
echo "   enabling AlwaysDefrag.."
echo "1" > /proc/sys/net/ipv4/ip_always_defrag


# Dynamic IP users:
#
#   If you get your Internet IP address dynamically from SLIP, PPP, or DHCP,
#   enable the following option.  This enables dynamic-ip address hacking in
#   IP MASQ, making the life with DialD, PPPd, and similar programs much easier.
#
#echo "   enabling DynamicAddr.."
#echo "1" > /proc/sys/net/ipv4/ip_dynaddr


#Clearing any previous configuration
#
#  Unless specified, the defaults for INPUT and OUTPUT is ACCEPT
#    The default for FORWARD is REJECT
#
#   Isn't ACCEPT insecure?  To some degree, YES, but this is our testing
#   phase.  Once we know that IPMASQ is working well, I recommend you run
#   the rc.firewall-*-stronger rulesets which set the defaults to DROP but
#   also include the critical additional rulesets to still let you connect to
#   the IPMASQ server, etc.
#
echo "   clearing any existing rules and setting default policy.."
/sbin/ipfwadm -I -p accept
/sbin/ipfwadm -O -p accept
/sbin/ipfwadm -F -p reject
/sbin/ipfwadm -I -f
/sbin/ipfwadm -O -f
/sbin/ipfwadm -F -f


# MASQ timeouts
```

```
#
#   2 hrs timeout for TCP session timeouts
#  10 sec timeout for traffic after the TCP/IP "FIN" packet is received
#  160 sec timeout for UDP traffic (Important for MASQ'ed ICQ users)
#
echo "   setting default timers.."
/sbin/ipfwadm -M -s 7200 10 160


# DHCP:  For people who receive their external IP address from either DHCP or
#        BOOTP such as DSL or Cablemodem users, it is necessary to use the
#        following before the deny command.
#
#        This example is currently commented out.
#
#
#/sbin/ipfwadm -I -a accept -S 0/0 67 -D 0/0 68 -W $EXTIF -P udp


# Enable simple IP forwarding and Masquerading
#
#  NOTE:  The following is an example for an internal LAN address in the
#         192.168.0.x network with a 255.255.255.0 or a "24" bit subnet mask
#         connecting to the Internet on interface eth0.
#
#         ** Please change this network number, subnet mask, and your Internet
#         ** connection interface name to match your internal LAN setup.
#
echo "   enabling IPMASQ functionality on $EXTIF"
/sbin/ipfwadm -F -p deny
/sbin/ipfwadm -F -a m -W $EXTIF -S $INTLAN -D 0.0.0.0/0

echo -e "\nrc.firewall-ipfwadm v$FWVER done.\n"
```

<rc.firewall-ipfwadm STOP>

Once you are finished with editing the /etc/rc.d/rc.firewall-ipfwadm ruleset, make it executable by typing in "`chmod 700 /etc/rc.d/rc.firewall-ipfwadm`"

Now that the firewall ruleset is ready to go, you need to let it run after every reboot. You could either do this by running it by hand everytime (such a pain) or add it to the boot scripts. We have covered two methods below: Redhat (SyS-V style) and Slackware (BSD style)

Redhat and Redhat-derived distros:

- There are two ways to automatically load things in Redhat: /etc/rc.d/rc.local or a init script in /etc/rc.d/init.d/. The first method is the easiest but isn't doing it the Sys-V way. All you have to do is add the line:

```
echo "Loading the rc.firewall-ipfwadm ruleset.."

/etc/rc.d/rc.firewall-ipfwadm
```

The problem with this approach is that the firewall isn't executed until the last stages of booting. The preferred approach is to have the firewall loaded just after the networking subsystem is loaded. To do this, copy the following file into the /etc/rc.d/init.d directory:

<firewall-ipfwadm START>

```
#!/bin/sh
#
# chkconfig: 2345 11 89
#
# description: Loads the rc.firewall-ipfwadm ruleset.
#
# processname: firewall-ipfwadm
# pidfile: /var/run/firewall.pid
# config: /etc/rc.d/rc.firewall-ipfwadm
# probe: true

# ------------------------------------------------------------------------
# v02/09/02
#
# Part of the copyrighted and trademarked TrinityOS document.
# http://www.ecst.csuchico.edu/~dranch
#
# Written and Maintained by David A. Ranch
# dranch@trinnet.net
#
# Updates
# -------
#
# ------------------------------------------------------------------------


# Source function library.
. /etc/rc.d/init.d/functions

# Check that networking is up.

# This line no longer work with bash2
#[ ${NETWORKING} = "no" ] && exit 0
# This should be OK.
[ "XXXX${NETWORKING}" = "XXXXno" ] && exit 0

[ -x /sbin/ifconfig ] || exit 0

# The location of various iptables and other shell programs
#
#   If your Linux distribution came with a copy of iptables, most
#   likely it is located in /sbin.  If you manually compiled
#   iptables, the default location is in /usr/local/sbin
#
# ** Please use the "whereis iptables" command to figure out
# ** where your copy is and change the path below to reflect
# ** your setup
#
```

```
IPFWADM=/sbin/ipfwadm


# See how we were called.
case "$1" in
  start)
    /etc/rc.d/rc.firewall-ipfwadm
    ;;

  stop)
    echo -e "\nFlushing firewall and setting default policies to REJECT\n"

    $IPFWADM -I -p REJECT
    $IPFWADM -O -p REJECT
    $IPFWADM -F -p REJECT

    $IPFWADM -I -f
    $IPFWADM -O -f
    $IPFWADM -F -f
    ;;

  restart)
    $0 stop
    $0 start
    ;;

  status)
    $IPFWADM -l
    ;;

  mlist)
    $IPFWADM -M -l
    ;;

  *)
    echo "Usage: firewall-ipfwadm {start|stop|status|mlist}"
    exit 1
esac

exit 0
```

<firewall-ipfwadm STOP>

With this script in place, all you need to do now is make it executable and then make it load upon reboot. First, make it executable by running:

```
#Redhat-style
#
chmod 700 /etc/rc.d/init.d/firewall-ipfwadm
```

Now, make the ruleset load upon reboot:

```
#Redhat style
#
chkconfig --level=345 firewall-ipfwadm on
```

That's it! Now upon boot, the firewall will be loaded automatically. Just to make sure, run the command to see that the firewall should start upon reboot by running the command:

```
#Redhat style
#
chkconfig --list firewall-ipfwadm

#The output should look like:
#
firewall-ipfwadm 0:off 1:off 2:off 3:on 4:on 5:on 6:off
```

Slackware:

- There are two ways to automatically load things in Slackware: /etc/rc.d/rc.local or editing the /etc/rc.d/rc.inet2 file. The first method is the easiest but isn't the most secure (see below). All you have to do is append the following lines to the /etc/rc.d/rc.local file:

```
echo "Loading the rc.firewall-ipfwadm ruleset.."

/etc/rc.d/rc.firewall-ipfwadm
```

The problem with this approach is that if you are running a STRONG firewall ruleset, the firewall isn't executed until the last stages of booting. The preferred approach is to have the firewall loaded just after the networking subsystem is loaded. For now, the HOWTO only covers how to do so using /etc/rc.d/rc.local but if you know what you're doing (it's easy), go ahead and and modify the inet2 startup script to load the /etc/rc.d/rc.firewall-ipfwadm file just after the network is up. If you want a more detailed guide and/or a stronger firewall ruleset, I recommend you check out Section 10 of TrinityOS found in the links section at the bottom of this HOWTO.

Notes on how users might want to change the above firewall ruleset:

You could have also enabled IP Masquerading on a PER MACHINE basis instead of the above method enabling an ENTIRE TCP/IP network. For example, say if I wanted only the 192.168.0.2 and 192.168.0.8 hosts to have access to the Internet and NOT any of the other internal machines. I would change the in the "Enable simple IP forwarding and Masquerading" section (shown above) of the /etc/rc.d/rc.firewall-ipfwadm ruleset.

```
# Enable simple IP forwarding and Masquerading
# v2.01
#
#  NOTE:  The following is an example to only allow IP Masquerading for the
#         192.168.0.2 and 192.168.0.8 machines with a 255.255.255.0 or a "24"
#         bit subnet mask connected to the Internet on interface eth0.
#
#         ** Please change this network number, subnet mask, and your Internet
#         ** connection interface name to match your internal LAN setup
#
#         Please use the following in ADDITION to the simple rulesets above
for
```

```
#         specific MASQ networks.
#
/sbin/ipfwadm -F -p deny
/sbin/ipfwadm -F -a m -W $EXTIF -S 192.168.0.2/32 -D 0.0.0.0/0
/sbin/ipfwadm -F -a m -W $EXTIF -S 192.168.0.8/32 -D 0.0.0.0/0
```

Common mistakes:

What appears to be a common mistake with new IP Masq users is to make the first command:

```
ipfwadm -F -p masquerade
```

Do **NOT** make your default policy MASQUERADING. Otherwise, someone who has the ability to manipulate their routing tables will be able to tunnel straight back through your gateway, using it to masquerade their OWN identity!

Again, you can add these lines to the `/etc/rc.d/rc.firewall-ipfwadm` file, one of the other rc files (if you prefer), or manually add those lines every time you need IP Masquerade.

Please see Section 6.4.3 and Section 6.4.3for a detailed guide and stronger examples of IPCHAINS and IPFWADM ruleset examples.

Chapter 4. Configuring the other internal to-be MASQed machines

Besides setting the appropriate IP address for each internal MASQed machine (either statically or though DHCP), you should also set each internal machine with the appropriate gateway IP address of the Linux MASQ server and required DNS servers. In general, this is rather straight forward. You simply enter the address of your Linux host (192.168.0.1 is used throughout this HOWTO) as the machine's gateway address.

For the Domain Name Service (DNS), you add in any DNS servers that are available to you to use. The most apparent one(s) should be the DNS servers that your Linux server uses. You can optionally add any "domain search" suffix as well for quicker connections, etc.

After you have properly reconfigured the internal MASQed machines, remember to restart their appropriate network services or reboot them if need be.

The following configuration instructions assume that you are using a Class C network with 192.168.0.1 as your Linux MASQ server's address. Please note that 192.168.0.0 and 192.168.0.255 are reserved TCP/IP address per RFC1918 for uses just like enabling IP Masquerade services.

As it stands, the following Platforms have been tested as internal MASQed machines. This is only an EXAMPLE of all compatible OSes out there:

- Apple Macintosh OS and OS-X (with MacTCP or Open Transport or the BSD TCP/IP stack)
- AT&T Unix (Caldera)
- *BSD systems including Free/Net/Open/BSDi/386/etc.
- Commodore Amiga (with AmiTCP or AS225-stack)
- Digital VAX Stations 3520 and 3100 with UCX (TCP/IP stack for VMS)

- Digital Ultrix, Digital Unix (Compaq Tru/64)
- HP HP/UX
- IBM AIX running on RS/6000, PowerPC, etc.
- IBM OS/2 (including Warp v3)
- IBM OS400 running on a AS/400
- Linux distributions from vendors like Caldera, Corel, Debian, Mandrake, Redhat, Slackware, SuSe, etc. running various kernels like 1.2.x, 1.3.x, 2.0.x, 2.1.x, 2.2.x, 2.3.x, 2.4.x, etc.
- Microsoft DOS (with NCSA Telnet package, DOS Trumpet works partially)
- Microsoft Windows 3.1 (with the Netmanage or FTP packages)
- Microsoft Windows For Workgroup 3.11 (with a TCP/IP package)
- Microsoft Windows 95, OSR2, 98, 98se, Me
- Microsoft Windows NT 3.51, 4.0, 2000, XP - (both workstation (professional) and server versions)
- Novell Netware 4.01, 5.x, etc. with the TCP/IP service
- SCO Openserver (v3.2.4.2 and 5) and UnixWare (AT&T Unix)
- Sun Solaris 2.51, 2.6, 7, 8
- heheh.. what else am I missing?

4.1. Configuring Microsoft Windows 95 and OSR2

1. ** Please note that some prompts might be different based upon the build version of Windows95 you are running **

   If you haven't installed your network card and adapter driver, do so now. Descriptions to perform this step is beyond the scope of this document and though it is fairly simple, if you haven't done this before, please seek assitance.
2. Go to the **'Control Panel'** --> **'Network'**.
3. Click on **Add** --> **Protocol** --> **Manufacture: Microsoft** --> **Protocol: 'TCP/IP protocol'** if you don't already have it installed.
4. Highlight the TCP/IP item bound to your correct Windows95 network card e.g. (TCP/IP --> Intel EtherExpress Pro/100+) and select '**Properties**'. Here, you have two options: configure a static address or use DHCP. Static addresses are simple but require that you NEVER configure duplication IPs on different machines. The alternative is DHCP which automatically configures all DHCP-enabled workstations things like IP addresses, DNS servers, etc. from a central server (typically the Linux MASQ server).

   DHCP enabled:

   To use DHCP, simply click on the "Use DHCP to assign addresses" button. Please note that configuring a DHCP server is beyond the scope of this HOWTO but it is fully covered in TrinityOS and other Linux HOWTOs.

   Static Addresses:

   Now goto the **'IP Address'** tab and set IP Address to 192.168.0.x, (1 < x < 255), and set the Subnet Mask to 255.255.255.0
5. Now select the **"Gateway"** tab and add 192.168.0.1 as your gateway under **'Gateway'** and hit "Add".
6. Under the **'DNS Configuration'** tab, make sure to put enter in a name for this machine and specify your official domain name. If you don't have your own domain, enter in the domain of your ISP. Next, you need to specify the DNS servers you plan on using.

   DHCP: No entries are required as this is configured dynamically via DHCP.

   STATIC: Add all of the DNS servers that your Linux MASQ server uses (usually found in `/etc/resolv.conf`). Usually these DNS servers are located at your ISP though you could be running either your own Caching or Authoritative DNS server on your Linux MASQ server as well. Again, setting up DNS services is beyond the scope of this HOWTO but it is covered by TrinityOS as well as the LDP's DNS HOWTO.

Optionally, you can add any appropriate domain search suffixes as well. This allows users to simply type in the hostname of the destination computer instead of the fully qualified domain name (FQDN). This is similar to the PATH function for finding common Unix commands.

7. Leave all of the other settings alone as they are unless (even dangerous) if you don't know what you're doing.
8. Click **'OK'** in all dialog boxes and restart your system.
9. As an initial test, `Ping` the Linux MASQ server to test the network connection: **'Start/Run'**, type: `ping 192.168.0.1` (This is only an INTERNAL LAN connection test, you might not be able to `ping` the outside world yet.) If you don't see "replies" to your PINGs, please verify your network configuration.
10. You can optionally create a `HOSTS` file in the C:\Windows directory so that you can ping the "hostname" of the machines on your LAN without the need for a DNS server. There is an example called `HOSTS.SAM` in the C:\windows directory for an example.

4.2. Configuring Windows NT

1. If you haven't installed your network card and adapter driver, do so now. Descriptions to perform this task is beyond the scope of this document.
2. Go to **'Control Panel'** --> **'Network'** --> **Protocols**
3. Add the TCP/IP Protocol and related Components from the **'Add Software'** menu if you don't have TCP/IP service installed already.
4. Under **'Network Software and Adapter Cards'** section, highlight the **'TCP/IP Protocol'** in the **'Installed Network Software'** selection box.
5. In **'TCP/IP Configuration'**, select the appropriate adapter, e.g. `[1]Intel EtherExpress Pro/100+`. Then set the IP Address to 192.168.0.x (1 < x < 255), then set the Subnet Mask to 255.255.255.0 and Default Gateway to 192.168.0.1.
6. Do not enable any of the following options (unless you know what you are doing):

    - **'Automatic DHCP Configuration'** : Unless you have a DHCP server running on your network.
    - Put anything in the **'WINS Server'** input areas : Unless you have setup one or more WINS servers.
    - **Enable IP Forwardings** : Unless you are routing on your NT machine and really -REALLY- know EXACTLY what you're doing.

7. Click **'DNS'**, fill in the appropriate information that your Linux host uses (usually found in /etc/resolv.conf) and then click **'OK'** when you're done.
8. Click **'Advanced'**, be sure to DISABLE **'DNS for Windows Name Resolution'** and **'Enable LMHOSTS lookup'** unless you known what these options do. If you want to use a LMHOSTS file, it is stored in C:\winnt\system32\drivers\etc.
9. Click **'OK'** on all dialog boxes and restart the system.
10. `As an initial test, ping` the Linux MASQ server to test the network connection: **'File/Run'**, type: `ping 192.168.0.1`(This is only an INTERNAL LAN connection test, you you might not be able to `ping` the outside world yet.) If you don't see any "replies" to your PINGs, please verify your network configuration.

4.3. Configuring Windows for Workgroup 3.11

1. If you haven't installed your network card and adapter driver, do so now. Descriptions to perform this task is beyond the scope of this document.
2. Install the TCP/IP 32b package if you haven't already.
3. In **'Main'/'Windows Setup'/'Network Setup'**, click on **'Drivers'**.
4. Highlight **'Microsoft TCP/IP-32 3.11b'** in the **'Network Drivers'** section, click **'Setup'**.
5. Set the IP Address to 192.168.0.x (1 < x < 255), then set the Subnet Mask to 255.255.255.0 and Default Gateway to

192.168.0.1

6. Do not enable any of the following options (unless you know what you are doing):

    - '**Automatic DHCP Configuration**' : Unless you have a DHCP server running on your network.
    - Put anything in the '**WINS Server**' input areas : Unless you have setup one or more WINS servers.

7. Click '**DNS**', fill in the appropriate information your Linux host uses (usually found in /etc/resolv.conf). Then click '**OK**' when you're done with it.
8. Click '**Advanced**', check '**Enable DNS for Windows Name Resolution**' and '**Enable LMHOSTS lookup**' found in c:\windows.
9. Click '**OK**' in all dialog boxes and restart the system.
10. As an initial test, `ping` the linux box to test the network connection: '**File/Run**', type: `ping 192.168.0.1` (This is only an INTERNAL LAN connection test so you might not be able to `ping` the outside world yet.) If you don't see "replies" to your PINGs, please verify your network configuration.

4.4. Configuring UNIX Based Systems

1. If you haven't installed your network card and either re-configured the network subsystem or recompiled your kernel with the appropriate adapter driver, do so now. Descriptions to perform this task is beyond the scope of this document but are covered in the Networking HOWTO.
2. Install TCP/IP networking, such as the net-tools package, if you don't have it already.
3. Set **IPADDR** to 192.168.0.x (1 < x < 255), then set **NETMASK** to 255.255.255.0, **GATEWAY** to 192.168.0.1, and **BROADCAST** to 192.168.0.255.

    - Redhat (Mandrake / TurboLinux / etc): You can edit the `/etc/sysconfig/network-scripts/ifcfg-eth0` file, or simply do so through the Control Panel (Linuxconf).
    - Slackware: You need to edit the /etc/rc.d/rc.inet1 file to configure the network subsystem.
    - To Add: Debian, Suse, Caldera, etc. Please email dranch@trinnet.net if you can tell me what distro uses what files to configure the networking subsystem.

    Beyond this, most Linux distributions use significantly different network configuration mechanisms let alone other UNIXes such as SunOS, BSDi, Solaris, AIX, TruUnix, FreeBSD, etc.). Please refer to your specific UNIX documentation for more details.
4. Add your domain name service (DNS) and domain search suffix in `/etc/resolv.conf` and for the appropreiate UNIX versions, edit the /etc/nsswitch.conf file to enable DNS services.
5. You may also want to update your `/etc/networks` file depending on your version of UNIX and the system's settings.
6. Restart the appropriate services, or simply restart your system.
7. As an initial test, run the `ping` command: `ping 192.168.0.1` to test the connection to your gateway machine. (This is only an INTERNAL LAN connection test, so you might not be able to `ping` the outside world yet.) If you don't see "replies" to your PINGs, please verify your network configuration.

4.5. Configuring DOS using NCSA Telnet package

1. If you haven't installed your network card, do so now. Descriptions to perform this task is beyond the scope of this document.
2. Load the appropriate packet driver. For example: an NE2000 Ethernet card set for I/O port 300 and IRQ 10, would need to be issued `nwpd 0x60 10 0x300`
3. Make a new directory, and then unpack the NCSA Telnet package: `pkunzip tel2308b.zip`
4. Use a text editor to open the `config.tel` file

5. Set `myip=192.168.0.x` (1 < x < 255), and netmask=255.255.255.0
6. In this example, you should set `hardware=packet, interrupt=10, ioaddr=60`
7. You should have at least one individual machine specified as the gateway, i.e. the Linux host:

```
name=default
host=yourlinuxhostname
hostip=192.168.0.1
gateway=1
```

8. Have another specified as a domain name service:

```
name=dns.domain.com ; hostip=123.123.123.123; nameserver=1
```

   Note: substitute the appropriate information about the DNS according what your Linux host uses
9. Save your `config.tel` file
10. As an initial test, `ping` the Linux MASQ server to test the network connection: `ping 192.168.0.1` If you don't receive any replies, please verify your network configuration.

4.6. Configuring MacOS Based System Running MacTCP

1. If you haven't installed the appropriate driver software for your Ethernet adapter, do so now. Descriptions to perform this task is beyond the scope of this document.
2. Open the **MacTCP control panel**. Select the appropriate network driver (Ethernet, NOT EtherTalk) and click on the '**More...**' button.
3. Under '**Obtain Address:**', click '**Manually**'.
4. Under '**IP Address:**', select **class C** from the popup menu. Ignore the rest of the dialog box sections.
5. Fill in the appropriate information under '**Domain Name Server Information:**'.
6. Under '**Gateway Address:**', enter 192.168.0.1
7. Click '**OK**' to save the settings. In the main window of the **MacTCP control panel**, enter the IP address of your Mac (192.168.0.x, 1 < x < 255) in the '**IP Address:**' box.
8. Close the **MacTCP control panel**. If a dialog box pops up, notifying you to do so, then restart the system.
9. You may optionally ping the Linux box to test the network connection. If you have the freeware program **MacTCP Watcher** , click on the '**Ping**' button, and enter the address of your Linux box (192.168.0.1) in the dialog window that pops up. (This is only an INTERNAL LAN connection test, you can't ping the outside world yet.) If you don't see "replies" to your PINGs, please verify your network configuration.
10. You can optionally create a `Hosts` file in your System Folder so that you can use the hostnames of the machines on your LAN. The file should already exist in your System Folder, and should contain some (commented-out) sample entries which you can modify according to your needs.

4.7. Configuring MacOS Based System Running Open Transport

1. If you haven't installed the appropriate driver software for your Ethernet adapter, do so now. Descriptions to perform this task is beyond the scope of this document.

2. Open the **TCP/IP Control Panel** and choose '**User Mode ...**' from the **Edit** menu. Make sure the user mode is set to at least '**Advanced**' and click the '**OK**' button.
3. Choose '**Configurations...**' from the **File** menu. Select your '**Default**' configuration and click the '**Duplicate...**' button. Enter 'IP Masq' (or something to let you know that this is a special configuration) in the '**Duplicate Configuration**' dialog, it will probably say something like '**Default copy**'. Then click the '**OK**' button, and the '**Make Active**' button
4. Select '**Ethernet**' from the '**Connect via:**' pop-up.
5. Select the appropriate item from the '**Configure:**' pop-up. If you don't know which option to choose, you probably should re-select your '**Default**' configuration and quit. I use '**Manually**'.
6. Enter the IP address of your Mac (192.168.0.x, 1 < x < 255) in the '**IP Address:**' box.
7. Enter 255.255.255.0 in the '**Subnet mask:**' box.
8. Enter 192.168.0.1 in the '**Router address:**' box.
9. Enter the IP addresses of your domain name servers in the '**Name server addr.:**' box.
10. Enter the name of your Internet domain (e.g. 'microsoft.com') in the '**Starting domain name**' box under '**Implicit Search Path:**'.
11. The following procedures are optional. Incorrect values may cause erratic behavior. If you're not sure, it's probably better to leave them blank, unchecked and/or un-selected. Remove any information from those fields, if necessary. As far as I know, there is no way to use the TCP/IP dialogs to tell the system not to use a previously selected alternate "Hosts" file. If you know, I would be interested.

    Check the '**802.3**' if your network requires 802.3 frame types.
12. Click the '**Options...**' button to make sure that the TCP/IP is active. I use the '**Load only when needed**' option. If you continuously run and quit TCP/IP applications without rebooting your machine, you may find that unchecking the '**Load only when needed**' option will prevent/reduce the effects on your machine's memory management. With the item unchecked, the TCP/IP protocol stacks are always loaded and available for use. If checked, the TCP/IP stacks are automatically loaded when needed and un-loaded when not. It's the loading and unloading process that can cause your machine's memory to become fragmented.
13. You may ping the Linux box to test the network connection. If you have the freeware program **MacTCP Watcher**, click on the '**Ping**' button, and enter the address of your Linux box (192.168.0.1) in the dialog box that pops up. (This is only an INTERNAL LAN connection test, you can't ping the outside world yet.) If you don't see "replies" to your PINGs, please verify your network configuration.
14. You can optionally create a `Hosts` file in your System Folder so that you can use the hostnames of the machines on your LAN. The file may or may not already exist in your System Folder. If so, it should contain some (commented-out) sample entries which you can modify according to your needs. If not, you can get a copy of the file from a system running MacTCP, or just create your own (it follows a subset of the Unix `/etc/hosts` file format, described on RFC1035). Once you've created the file, open the **TCP/IP control panel**, click on the '**Select Hosts File...**' button, and open the `Hosts` file.
15. Click the close box or choose '**Close**' or '**Quit**' from the **File** menu, and then click the '**Save**' button to save the changes you have made.
16. The changes take effect immediately, but rebooting the system won't hurt.

4.8. Configuring Novell network using DNS

1. If you haven't installed the appropriate driver software for your Ethernet adapter, do so now. Descriptions to perform this task is beyond the scope of this document.
2. Downloaded tcpip16.exe from The Novell LanWorkPlace page
3.

```
edit c:\nwclient\startnet.bat: (here is a copy of mine)
SET NWLANGUAGE=ENGLISH
LH LSL.COM
LH KTC2000.COM
LH IPXODI.COM
LH tcpip
LH VLM.EXE
F:
```

4.

```
edit c:\nwclient\net.cfg: (change link driver to yours i.e. NE2000)
Link Driver KTC2000
        Protocol IPX 0 ETHERNET_802.3
        Frame ETHERNET_802.3
        Frame Ethernet_II
        FRAME Ethernet_802.2

NetWare DOS Requester
           FIRST NETWORK DRIVE = F
           USE DEFAULTS = OFF
           VLM = CONN.VLM
           VLM = IPXNCP.VLM
           VLM = TRAN.VLM
           VLM = SECURITY.VLM
           VLM = NDS.VLM
           VLM = BIND.VLM
           VLM = NWP.VLM
           VLM = FIO.VLM
           VLM = GENERAL.VLM
           VLM = REDIR.VLM
           VLM = PRINT.VLM
           VLM = NETX.VLM

Link Support
        Buffers 8 1500
        MemPool 4096

Protocol TCPIP
        PATH SCRIPT     C:\NET\SCRIPT
        PATH PROFILE    C:\NET\PROFILE
        PATH LWP_CFG    C:\NET\HSTACC
        PATH TCP_CFG    C:\NET\TCP
        ip_address      192.168.0.xxx
        ip_router       192.168.0.1


Change the IP address in the above "ip_address" field (192.168.0.x, 1 <
x
< 255) and finally create c:\bin\resolv.cfg:

SEARCH DNS HOSTS SEQUENTIAL
NAMESERVER xxx.xxx.xxx.xxx
```

```
NAMESERVER yyy.yyy.yyy.yyy
```

5. Now edit the above "NAMESERVER" entries and replace them with the correct IP addresses for your local DNS server.
6. Issue a `ping` command: `ping 192.168.0.1` to test the connection to your gateway machine. (This is only an INTERNAL LAN connection test, you can't `ping` the outside world yet.) If you don't see "replies" to your PINGs, please verify your network configuration.

4.9. Configuring OS/2 Warp

1. If you haven't installed the appropriate driver software for your Ethernet adapter, do so now. Descriptions to perform this task is beyond the scope of this document.
2. Install the TCP/IP protocol if you don't have it already.
3. Go to **Programs/TCP/IP (LAN) / TCP/IP** Settings
4. In '**Network**', add your TCP/IP Address (192.168.0.x) and set your netmask (255.255.255.0)
5. Under '**Routing**', press '**Add**'. Set the **Type** to '**default**' and type the IP Address of your Linux Box in the Field '**Router Address**'. (192.168.0.1).
6. Set the same DNS (Nameserver) Address that your Linux host uses in '**Hosts**'.
7. Close the TCP/IP control panel. Say yes to the following question(s).
8. Reboot your system
9. You may ping the Linux box to test the network configuration. Type `'ping 192.168.0.1'` in a 'OS/2 Command prompt Window'. When ping packets are received all is ok.

4.10. Configuring OS/400 on a IBM AS/400

The descriptions to configure TCP/IP on OS/400 version V4R1M0 running on a AS/400 is beyond the scope of this document.

1) To perform any communications configuration tasks on your AS/400, you must have the special authority of *IOSYSCFG (I/O System Configuration) defined in your user profile. You can check the characteristics of your user profile with the DSPUSRPRF command.

2) Type GO CFGTCP command th reach the Configure TCP/IP menu.

3) Select Option 2 - Work with TCP/IP Routes.

4) Enter a 1 on the Opt field to add a route. * In Route Destination type *DFTROUTE * In Subnet Mask type *NONE * In Type of Service type *NORMAL * In Nex Hop type the address of your gataway (the Linux box)

4.11. Configuring Other Systems

The same logic should apply to setting up other platforms. Consult the sections above. If you're interested in writing about any of systems that have not been covered yet, please send a detail setup instruction to ambrose@writeme.com and dranch@trinnet.net.

Chapter 5. Testing IP Masquerade

Finally, it's time to give IP Masquerading an official try after all this hard work. If you haven't already rebooted your Linux box, do so to make sure the machines boots ok, executes the /etc/rc.d/rc.firewall-* ruleset, etc. Next, make sure that both the internal LAN connection and connection of your Linux hosts to the Internet is okay.

Follow these -11- tests to make sure all aspects of your MASQ setup is running properly:

5.1. Loading up the rc.firewall ruleset

Step One: run the correct firewall for your machine via command "/etc/rc.d/rc.firewall-[iptables / ipchains /ipfwadm]". For example, Linux 2.6 users would run "/etc/rc.d/rc.firewall-iptables"

Does it load with some strange errors? Here are some exmaples and help to fix them:

- Problem #1:

```
ip_tables, Using /lib/modules/2.4.2-2/kernel/net/ipv4/netfilter/ip_tables.o
/lib/modules/2.4.2-2/kernel/net/ipv4/netfilter/ip_tables.o: init_module: Device
or resource busy
Hint: insmod errors can be caused by incorrect module parameters, including
invalid IO or IRQ parameters
```

  Run the command "/sbin/lsmod" and make sure the module "ipchains.o" is NOT installed. If it is installed, your machine (most likely Redhat-7.x based) is probably trying to load an IPCHAINS ruleset which is incompatible with IPTABLES.

  To disable this from happening in the future, run the command:

```
   chkconfig --level=2345 ipchains off
```

  To remove the "ipchains" module without rebooting, run the command:

```
   /sbin/rmmod ipchains
```

  and the re-try to load the rc.firewall-* ruleset.

- Problem #2:

```
    .
    .
    Creating a DROP chain..
    iptables v1.2.3: log-level `info' ambiguous
    .
    .
```

Your version of IPTABLES it too old. You need to upgrade it with a newer version via an updated RPM, DEB, or via compiling up the source. You can get an updated version from your Linux distribution manufacturer or from the NetFilter WWW site. This is all covered in the Section 2.6 section.

- Problem #3:

```
    No such file:
```

Did you copy this rc.firewall-* file from a DOS machine? Load the rc.firewall-* file in a binary editor such as ViM (vim -b /etc/rc.d/rc.firewall-*) and make sure that every line is NOT finished with a ^M.

5.2. Testing internal MASQ client PC connectivity

- **Step Two: Testing internal MASQ client PC connectivity**

From an internal MASQed computer, try pinging its local IP address (i.e. **ping 192.168.0.10**). This will verify that TCP/IP is correctly working on the local machine. Almost ALL modern operating systems have built-in support for the "ping" command. If this ping doesn't work, make sure that TCP/IP is correctly configured on the MASQed PC as described earlier in Chapter 4 of this HOWTO. The output should look something like the following (hit Control-C to abort the ping):

```
-----------------------------------
masq-client# ping 192.168.0.10
PING 192.168.0.10 (192.168.0.10): 56 data bytes
64 bytes from 192.168.0.10: icmp_seq=0 ttl=255 time=0.8 ms
64 bytes from 192.168.0.10: icmp_seq=1 ttl=255 time=0.4 ms
64 bytes from 192.168.0.10: icmp_seq=2 ttl=255 time=0.4 ms
64 bytes from 192.168.0.10: icmp_seq=3 ttl=255 time=0.5 ms
^C

--- 192.168.0.10 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.5/0.8 ms
-----------------------------------
```

5.3. Testing internal MASQ client to MASQ server connectivity

- **Step Three: Testing internal MASQ client to MASQ server connectivity**

Next, from the same internal MASQed computer, try pinging the the IP address of the Linux MASQ server's INTERNAL interface (i.e. **ping 192.168.0.1**). This will verify that TCP/IP is correctly working on both the local and Linux MASQ machine. Almost ALL modern operating systems have built-in support for the "ping" command. If this ping doesn't work, make sure that TCP/IP is correctly configured on the MASQed Server as described by the various Network HOWTOs (URLs can be found in the requirements section for your 2.4.x kernel in Section 2.6, 2.2.x kernel in Section 2.7, or 2.0.x kernel in Section 2.8). The output should look something like the following (hit Control-C to abort the ping):

```
------------------------------------
masq-client# ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1): 56 data bytes
64 bytes from 192.168.0.1: icmp_seq=0 ttl=255 time=0.8 ms
64 bytes from 192.168.0.1: icmp_seq=1 ttl=255 time=0.4 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=255 time=0.4 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=255 time=0.5 ms
^C

--- 192.168.0.1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.5/0.8 ms
------------------------------------
```

If the ping failed, check the network connection between the MASQ server and the PC. If it's a DIRECT Ethernet connection (no hub or switch), you MUST have a "Ethernet cross-over cable". These cables are common and can be found at any computer store. Without this cable, the NICs (network cards) will not give you a "LINK" light. If you are using a hub or switch, make sure the ports connected to the MASQ server and MASQed client machine have a LINK light. If they do and the pings STILL don't work or there is a lot of packet loss, try different ports on the hub/switch (it not all that uncommon to have hub/switch ports die). Finally, if things still don't work perfectly, try replacing each of the NICs in the machines. You would be surprised how many people I've helped have found that their NIC cards were going bad and caused them all kinds of grief.

5.4. Testing internal MASQ server connectivity

- **Step Four: Testing internal MASQ server connectivity**

On the MASQ server, ping the internal IP address of the MASQ server's network interface card (i.e. **ping 192.168.0.1**). If this ping doesn't work, make sure that TCP/IP is correctly configured on the MASQed Server as described by the various Network HOWTOs (URLs can be found in the requirements section for your 2.4.x kernel in Section 2.6, 2.2.x kernel in Section 2.7, or 2.0.x kernel in Section 2.8). The output should look something like the following (hit Control-C to abort the ping):

```
-------------------------------------
masq-server# ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1): 56 data bytes
64 bytes from 192.168.0.1: icmp_seq=0 ttl=255 time=0.8 ms
64 bytes from 192.168.0.1: icmp_seq=1 ttl=255 time=0.4 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=255 time=0.4 ms
64 bytes from 192.168.0.1: icmp_seq=3 ttl=255 time=0.5 ms
^C

--- 192.168.0.1 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.5/0.8 ms
-------------------------------------
```

5.5. Testing internal MASQ server to MASQ client connectivity

- **Step Five: Testing internal MASQ server to MASQ client connectivity**

  Next from MASQed server, try pinging the IP address of one of the internal MASQ client computers (i.e. **ping 192.168.0.10**). This will verify that TCP/IP is correctly working on both the local server machine and on the MASQ client machine. If this ping doesn't work, make sure that TCP/IP is correctly configured on the MASQed PC as described earlier in Chapter 4 of this HOWTO. Also be sure that the cabling is correct (Ethernet: the NICs connecting the internal MASQ PC and the MASQ server have the "link" light lit up). If the ping does work, the output should look something like the following (hit Control-C to abort the ping):

```
-------------------------------------
masq-server# ping 192.168.0.10
PING 192.168.0.10 (192.168.0.10): 56 data bytes
64 bytes from 192.168.0.10: icmp_seq=0 ttl=255 time=0.8 ms
64 bytes from 192.168.0.10: icmp_seq=1 ttl=255 time=0.4 ms
64 bytes from 192.168.0.10: icmp_seq=2 ttl=255 time=0.4 ms
64 bytes from 192.168.0.10: icmp_seq=3 ttl=255 time=0.5 ms
^C

--- 192.168.0.10 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.5/0.8 ms
-------------------------------------
```

5.6. Testing External MASQ server Internet connectivity

- **Step Six: Testing external MASQ server to Internet connectivity**

  From the MASQ server, ping the external IP address of the MASQ server's EXTERNAL network interface that is connected to the Internet. This address might be a Ethernet interface, a PPP interface, etc. connection to your ISP. If you don't know

what this external IP address is, run the Linux command **"/sbin/ifconfig"** on the MASQ server itself to get the Internet address. The output should look something like the following (we are looking for the IP address of eth0):

```
-----------------------------------
eth0      Link encap:Ethernet  HWaddr 00:08:C7:A4:CC:5B
          inet addr:12.13.14.15  Bcast:12.13.14.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:6108459 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5422798 errors:8 dropped:0 overruns:0 carrier:8
          collisions:4675 txqueuelen:100
          Interrupt:11 Base address:0xfcf0
-----------------------------------
```

As you can see from the above, the external IP address is "12.13.14.15" for this example. So, now that you have your IP address after running the "ipconfig" command, ping your external IP address. This will confirm that the MASQ server has full network connectivity. The output should look something like the following (hit Control-C to abort the ping):

```
-----------------------------------
masq-server# ping 12.13.14.15
PING 12.13.14.15 (12.13.14.15): 56 data bytes
64 bytes from 12.13.14.15: icmp_seq=0 ttl=255 time=0.8 ms
64 bytes from 12.13.14.15: icmp_seq=1 ttl=255 time=0.4 ms
64 bytes from 12.13.14.15: icmp_seq=2 ttl=255 time=0.4 ms
64 bytes from 12.13.14.15: icmp_seq=3 ttl=255 time=0.5 ms
^C

--- 12.13.14.15 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.5/0.8 ms
-----------------------------------
```

If either of these tests doesn't work, you need to go back and double check your network cabling and verify that the two network interfaces on the MASQ server are seen in "dmesg". An example of this output would be the following towards the END of the "dmesg" command:

```
-----------------------------------
.
.
PPP: version 2.3.7 (demand dialling)
TCP compression code copyright 1989 Regents of the University of California
PPP line discipline registered.
3c59x.c:v0.99H 11/17/98 Donald Becker
http://cesdis.gsfc.nasa.gov/linux/drivers/
vortex.html
eth0: 3Com 3c905 Boomerang 100baseTx at 0xfe80,  00:60:08:a7:4e:0e, IRQ 9
  8K word-wide RAM 3:5 Rx:Tx split, autoselect/MII interface.
  MII transceiver found at address 24, status 786f.
  Enabling bus-master transmits and whole-frame receives.
eth1: 3Com 3c905 Boomerang 100baseTx at 0xfd80,  00:60:97:92:69:f8, IRQ 9
  8K word-wide RAM 3:5 Rx:Tx split, autoselect/MII interface.
  MII transceiver found at address 24, status 7849.
```

```
  Enabling bus-master transmits and whole-frame receives.
Partition check:
 sda: sda1 sda2 < sda5 sda6 sda7 sda8 >
 sdb:
.
.
-------------------------------------
```

Also be sure that the cabling is correct (Ethernet: the NICs connecting the external MASQ server to your ISP has the "link" light lit up). Finally, make sure that TCP/IP is correctly configured on the MASQed Server as described by the various Network HOWTOs (URLs can be found in the requirements section for your 2.4.x kernel in Section 2.6, 2.2.x kernel in Section 2.7, or 2.0.x kernel in Section 2.8).

5.7. Testing internal MASQ client to external MASQ server connectivity

- **Step Seven: Testing internal MASQ client to external MASQ server connectivity**

From an internal MASQed computer, ping the IP address of the MASQ server's EXTERNAL TCP/IP address obtained in Step FIVE above. This address could be from your Ethernet, PPP, etc. interface which is ultimately the address connected to your ISP. This ping test will prove that Linux masquerading (ICMP Masquerading specifically) and IP forwarding is working.

If everthing thing is working correctly, the output should look something like the following (hit Control-C to abort the ping):

```
-------------------------------------
masq-client# ping 12.13.14.15
PING 12.13.14.15 (12.13.14.15): 56 data bytes
64 bytes from 12.13.14.15: icmp_seq=0 ttl=255 time=0.8 ms
64 bytes from 12.13.14.15: icmp_seq=1 ttl=255 time=0.4 ms
64 bytes from 12.13.14.15: icmp_seq=2 ttl=255 time=0.4 ms
64 bytes from 12.13.14.15: icmp_seq=3 ttl=255 time=0.5 ms
^C

--- 12.13.14.15 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.4/0.5/0.8 ms
-------------------------------------
```

If this test doesn't work, first make sure that the "Default Gateway" on the MASQed PC is pointing to the IP address on the MASQ -SERVERs- INTERNAL NIC. Also double check that the /etc/rc.d/rc.firewall-* script was run without any errors. Just as a test, try re-running the /etc/rc.d/rc.firewall-* script now to see if it runs OK. Also, though most kernels support it by default, make sure that you enabled "ICMP Masquerading" in the kernel comfiguration and "IP Forwarding" in your /etc/rc.d/rc.firewall-* script.

If you still can't get things to work, take a look at the output from the following commands run on the Linux MASQ SERVER:

- "**ifconfig**" : Make sure the interface for your Internet connection (be it ppp0, eth0, etc.) is UP and you have the correct IP address for the Internet connection. An example of this output is shown in STEP FIVE above.
- "**netstat -rn**" : Make sure your default gateway (the column with an IP address in the Gateway column) is set. An example of this output might look like:

```
------------------------------------
masq-server# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.0.1     0.0.0.0         255.255.255.255 UH        0 16384      0 eth1
12.13.14.15     0.0.0.0         255.255.255.255 UH        0 16384      0 eth0
12.13.14.0      0.0.0.0         255.255.255.0   U         0 0          0 eth0
192.168.0.0     0.0.0.0         255.255.255.0   U         0 0          0 eth1
127.0.0.0       0.0.0.0         255.0.0.0       U         0 16384      0 lo
0.0.0.0         12.13.14.1      0.0.0.0         UG        0 16384      0 eth0
------------------------------------
```

Notice the very LAST line that starts with 0.0.0.0? Notice that it also has an IP address in the "Gateway" field? You should specify an IP address for your specific setup in that field (this is typically done automatically when your Internet connection is enabled).

- "**cat /proc/sys/net/ipv4/ip_forward**" : Make sure it says "1" so that Linux forwarding is enabled
- Run the command "**/sbin/ipchains -n -L**" for 2.2.x users or "**/sbin/ipfwadm -F -l**" for 2.0.x users. Specifically, look for the FORWARDing section to make sure you have MASQ enabled. An example of an IPCHAINS output might look like for users using the SIMPLE rc.firewall-* ruleset:

```
------------------------------------
.
.
Chain forward (policy REJECT):
target     prot opt     source               destination          ports
MASQ       all  ------  192.168.0.0/24      0.0.0.0/0            n/a
ACCEPT     all  ----l-  0.0.0.0/0           0.0.0.0/0            n/a
.
.
------------------------------------
```

5.8. Testing external MASQ ICMP forwarding

- **Step Eight: Testing external MASQ ICMP forwarding**

From an internal MASQed computer, ping a static TCP/IP address (NOT a machine by DNS name) out on the Internet (i.e. **ping 152.2.210.80** (this technically the DNS name "metalab.unc.edu" which is home of MetaLabs' Linux Archive). If this works, it should look something like the result below and this ultimately shows that ICMP Masquerading is working properly. (hit Control-C to abort the ping):

```
-------------------------------------
masq-client# ping 152.2.210.80

PING 12.13.14.15 (152.2.210.80): 56 data bytes
64 bytes from 152.2.210.80: icmp_seq=0 ttl=255 time=133.4 ms
64 bytes from 152.2.210.80: icmp_seq=1 ttl=255 time=132.5 ms
64 bytes from 152.2.210.80: icmp_seq=2 ttl=255 time=128.8 ms
64 bytes from 152.2.210.80: icmp_seq=3 ttl=255 time=132.2 ms
^C

--- 152.2.210.80 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 128.8/131.7/133.4 ms
-------------------------------------
```

If this didn't work, again check your Internet connection. Make sure that the MASQ server itself can ping this address. If this works from the MASQ server, make sure you are using the simple rc.firewall-* ruleset and that you have ICMP Masqurading compiled into the Linux kernel.

Finally, make sure that the ruleset which enables IP MASQ is pointing to the correct EXTERNAL interface. PPPoE users should use the MASQ servers's logical PPP interface such as "ppp0" and /NOT/ the physical external interface like "eth0".

5.9. Testing MASQ functionality without DNS

- **Step Nine: Testing MASQ functionality without DNS**

Now try TELNETing to a remote IP address (i.e. **telnet 152.2.210.81** (this is technically the DNS name "metalab.unc.edu"). It might take a few seconds to get a login prompt since this is a VERY busy server. Did you get a login prompt like shown below? If so, it means that TCP Masquerading is running OK. If not, try TELNETing to some other hosts you think will support TELNET like 198.182.196.55 (www.linux.org). If this still doesn't work, make sure you are using the simple rc. firewall-* ruleset for this test. An example of this output might look like (hit Control-D to exit out of the TELNET):

```
--------------------------------
masq-client# telnet 152.2.210.80
Trying 152.2.210.80...
Connected to 152.2.210.80.
Escape character is '^]'.


SunOS 5.7


******************** Welcome to MetaLab.unc.edu ********************

 To login to MetaLab as a user, connect to login.metalab.unc.edu.
           This machine does not allow public telnet logins.

login: Connection closed by foreign host.
--------------------------------
```

5.10. Testing MASQ functionality with DNS resolution

- **Step Ten: Testing MASQ functionality with DNS resolution**

Now try TELNETing to a remote machine by DNS name (i.e. **"telnet metalab.unc.edu"** (IP address 152.2.210.81). If this works, the output should look like something below. With this test, this shows that UDP-based DNS is working fine.

```
--------------------------------
masq-client# telnet MetaLab.unc.edu
Trying 152.2.210.80...
Connected to 152.2.210.80.
Escape character is '^]'.


SunOS 5.7


******************** Welcome to MetaLab.unc.edu ********************

 To login to MetaLab as a user, connect to login.metalab.unc.edu.
           This machine does not allow public telnet logins.

login: Connection closed by foreign host.
--------------------------------
```

If this did not work, but Step EIGHT did work, make sure that you have one or more valid DNS servers configured on each of the MASQ CLIENT computer(s) as shown in Chapter 4. Please note that these DNS servers will typically be your ISP's DNS servers and NOT your local MASQ server. Some people might later choose to setup their OWN DNS servers but this is beyond the scope of this HOWTO.

5.11. Testing more MASQ functionality with DNS

- **Step Eleven: Testing more MASQ functionality with DNS**

As a last test, try browsing some **'INTERNET'** WWW sites on one of your **MASQ Client** machines, and see if you can reach them. For example, access the Linux Documentation Project site at http://www.tldp.org. If you are successful in bringing up that page, you can be fairly certain that everything is working FINE! If some WWW or FTP sites have problems, where other sites seem to work just fine, see the next step for more ideas.

If you see The Linux Documentation Project homepage, then **CONGRATULATIONS! It's working!** If the WWW site comes up correctly, then all other standard network tools such as PING, TELNET, SSH, and with their related IP MASQ modules loaded: FTP, Real Audio, IRC DCCs, Quake I/II/III, CuSeeme, VDOLive, etc. should work fine too! If FTP, IRC, RealAudio, Quake I/II/III, etc. aren't working or are performing poorly, verify that their associated Masquerading modules are loaded by running "lsmod" and also be sure you are loading the module with any non-default server ports. If you don't see your needed module, make sure your /etc/rc.d/rc.firewall-* script is loading them (i.e. remove the # character for a given IP

MASQ module).

5.12. Any remaining functional, performance, etc. issues...

- **Step Twelve: Any remaining functional, performance, etc. issues...**

  If your system passes all of the above tests, but functionality tests like WWW browsing, FTP, and other types of traffic aren't reliable or are slow, I recommend that you read the FAQ section of this HOWTO.. specifically the Section 7.15 FAQ entry. There might be other items in the FAQ section that will also help you as they have helped many other users in the past.

Chapter 6. Other IP Masquerade Issues and Software Support

6.1. Problems with IP Masquerade

Some TCP/IP application protocols will not currently work with Linux IP Masquerading because they either assume things about port numbers or encode TCP/IP addresses and/or port numbers in their data stream. These latter protocols need specific proxies or IP MASQ modules built into the masquerading code to make them work.

6.2. Incoming services

By default, Linux IP Masquerading cannot handle incoming services at all but there are a few ways that would allow this.

If you do not require high levels of security, then you can simply forward or redirect IP ports. There are various ways to perform this, though the most stable method is to use IPPORTFW. For more information, please see Section 6.7.

If you wish to have some level of authorization on incoming connections, then you will need to either configure TCP-wrappers or Xinetd to allow only specific IP addresses to pass. The TIS Firewall Toolkit is a good place to look for tools and information.

More details on incoming security can be found in the TrinityOS document and at IP Masquerade Resource.

6.3. Supported Client Software and Other Setup Notes

"**** The Linux Masquerade Application list has a lot of good information regarding applications that work through Linux IP masquerading. This site was recently taken over by Steve Grevemeyer, who implemented it with a full database backend. It's a great resource!** "

Generally, any application that uses standard TCP and UDP should work. If you have any suggestion, hints, etc., please see the IP Masquerade Resource for more details.

6.3.1. Network Clients that -Work- with IP Masquerade

General Clients:

Archie

all supported platforms, file searching client (not all archie clients are supported)

FTP

all supported platforms, with the **ip_masq_ftp.o** kernel module for active FTP connections.

Gopher client

all supported platforms

HTTP

all supported platforms, WWW surfing

IRC

all IRC clients on various supported platforms, DCC is supported via the **ip_masq_irc.o** module

NNTP (USENET)

all supported platforms, USENET news client

PING

all platforms, with ICMP Masquerading kernel option

POP3

all supported platforms, email clients

SSH

all supported platforms, Secure TELNET/FTP clients

SMTP

all supported platforms, email servers like Sendmail, Qmail, PostFix, etc.

TELNET

all supported platforms, remote session

TRACEROUTE

UNIX and Windows based platforms, some variations may not work

VRML

Windows(possibly all supported platforms), virtual reality surfing

WAIS client

all supported platforms

Multimedia and Communication Clients:

All H.323 programs

- MS Netmeeting, Intel Internet Phone Beta , and other H.323 applications - There are now two solutions to accomplish this through IPMASQed connections:

There is a stable BETA 2.2.x kernel module available on the MASQ WWW site or at http://www.coritel.it/projects/nat/implementation.htm to work with Microsoft Netmeeting v3.x code on 2.2.x kernels. There is also another module version on the MASQ WWW site specifically for Netmeeting 2.x with 2.0.x kernels, but this does not support Netmeeting v3.x.

Another commercial solution is the Equivalence's PhonePatch H.323 gateway.

Alpha Worlds

Windows, Client-Server 3D chat program

CU-SeeMe

all supported platforms, with the **ip_masq_cuseeme** module loaded, please see Section 6.8 for more details.

ICQ

all supported clients. Requires the Linux kernel to be either compiled with PORTFW support, have the ip_masq_icq module (2.2.x and 2.0.x only), or have a SOCKS proxy running. A full description of this configuration is in Section 6.9.

Internet Phone 3.2

Windows, Peer-to-peer audio communications, users can reach you only if you initiate the call, but those users cannot call you without a specific port forwarding setup. See Section 6.7for more details.

Internet Wave Player

Windows, network streaming audio

Powwow

Windows, Peer-to-peer Text audio whiteboard communications, users can reach you only if you initiate the call, but those users cannot call you without a specific port forwarding setup. See Section 6.7for more details.

Real Audio Player

Windows, network streaming audio, higher quality available with the **ip_masq_raudio** UDP module

True Speech Player 1.1b

Windows, network streaming audio

VDOLive

Windows, with the **ip_masq_vdolive** patch

Worlds Chat 0.9a

Windows, Client-Server 3D chat program

Games - See Section 6.10for more details on the LooseUDP patch

Battle.net

Works but requires TCP ports 116, 118 and UDP port 6112 IPPORTFWed to the client game machine. See Section 6.7for more details. Please note that FSGS and Bnetd servers still require IPPORTFW because they have not been re-written to be NAT-friendly.

BattleZone 1.4

Works with LooseUDP patch and new NAT-friendly -- email David Ranch for the .DLLs from Activision

Dark Reign 1.4

Works with LooseUDP patch or requires TCP ports 116 and 118 and UDP port 6112 IPPORTFWed to the game machine. See Section 6.7for more details.

Diablo

Works with LooseUDP patch or requires TCP ports 116 and 118 and UDP port 6112 IPPORTFWed to the game machine. Newer versions of Diablo use only TCP port 6112 and UDP port 6112. See Section 6.7for more details.

Heavy Gear 2

Works with LooseUDP patch or requires TCP ports 116 and 118 and UDP port 6112 IPPORTFWed to the game machine. See Section 6.7for more details.

Quake I/II/III

Works right out of the box but requires the **ip_masq_quake** module if there are more than one Quake I/II/III player behind a MASQ box. Also, this module only supports Quake I and QuakeWorld by default. If you need to support Quake II or non-default server ports, please see the module install section of Section 3.4.3 and Section 3.4.2 rulesets.

StarCraft

Works with the LooseUDP patch, IPPORTFWing TCP, and UDP ports 6112 to the internal MASQed game machine. See Section 6.7for more details.

WorldCraft

Works with LooseUDP patch

Other Clients:

Linux net-acct package

Linux, network administration-account package

NCSA Telnet 2.3.08

DOS, a suite containing telnet, ftp, ping, etc.

PC-anywhere for Windows

MS-Windows remotely controls a PC over TCP/IP, but only works if it is a client, but not a host without a specific port forwarding setup. See Section 6.7for more details.

Socket Watch

uses NTP - network time protocol

6.3.2. Clients that do not have full support in IP MASQ:

Intel Streaming Media Viewer Beta 1

Cannot connect to server

Netscape CoolTalk

Cannot connect to opposite side
WebPhone

Cannot work at present (it makes invalid assumptions about addresses).

6.4. Stronger firewall rulesets to run after initial testing

6.4.1. Stronger IP Firewall (IPTABLES) rulesets

<rc.firewall-iptables-stronger START>

```
#!/bin/sh
#
# rc.firewall-iptables-stronger
#
FWVER=0.88s

#          An example of a stronger IPTABLES firewall with IP Masquerade
#          support for 2.4.x kernels.
#
# Log:
#
#   0.88s - Updated the commands for dynamically addresses machines and
#           to point to an expanded FAQ section for more information
#
#   0.87s - Removed the unused drop-and-logit chain as it was only later
#           being deleted anyway
#   0.86s - Fixed a typo that had a preceeding ; instead of a #
#   0.85s - renamed from rc.firewall-2.4-stronger to rc.firewall-iptables-
#           stronger to reflect this script works for all IPTABLES enabled
#           platforms including 2.6.x kernels
#         - fixed an incorrect /24 netmask for the INTIP variable
#         - removed the unneeded SED variable
#   0.84s - Changed the defaults from 192.168.1.0 to 192.168.0.x to align
#           with the rest of the IPMASQ howto
#   0.83s - Added additional comments to make PORTFW configs more obvious
#   0.82s - Added a special ICMP filter to work around a Netfilter security
#           issue
#         - renamed the drop-and-log-it rule to reject-and-log-it
#   0.81s - Added an additional comment in the INPUT section for NOT
#           allowing all traffic in, but only select traffic
#   0.80s - Added a DISABLED ip_nat_irc kernel module section, changed the
#           default of the ip_conntrack_irc to NOT load by default, and
#           added additional kernel module comments
#   0.79s - ruleset now uses modprobe instead of insmod
#   0.78s - REJECT is not a legal policy yet; back to DROP
#   0.77s - Changed the default block behavior to REJECT not DROP
#   0.76s - Added a comment about the OPTIONAL WWW ruleset and a comment
#           where to put optional PORTFW commands
#   0.75s - Added clarification that PPPoE users need to use
#           "ppp0" instead of "eth0" for their external interface
```

```
#   0.74s - Changed the EXTIP command to work on NON-English distros
#   0.73s - Added comments in the output section that DHCPd is optional
#           and changed the default settings to disabled
#   0.72s - Changed the filter from the INTNET to the INTIP to be
#           stateful; moved the command VARs to the top and made the
#           rest of the script to use them
#   0.70s - Added a disabled examples for allowing internal DHCP
#           and external WWW access to the server
#   0.63s - Added support for the IRC module
#   0.62s - Initial version based upon the basic 2.4.x rc.firewall


echo -e "\nLoading rc.firewall-iptables-STRONGER - version $FWVER..\n"


# The location of various iptables and other shell programs
#
#   If your Linux distribution came with a copy of iptables, most
#   likely it is located in /sbin.  If you manually compiled
#   iptables, the default location is in /usr/local/sbin
#
# ** Please use the "whereis iptables" command to figure out
# ** where your copy is and change the path below to reflect
# ** your setup
#
#IPTABLES=/sbin/iptables
IPTABLES=/usr/local/sbin/iptables
#
LSMOD=/sbin/lsmod
DEPMOD=/sbin/depmod
MODPROBE=/sbin/modprobe
GREP=/bin/grep
AWK=/bin/awk
IFCONFIG=/sbin/ifconfig


#Setting the EXTERNAL and INTERNAL interfaces for the network
#
#  Each IP Masquerade network needs to have at least one
#  external and one internal network.  The external network
#  is where the natting will occur and the internal network
#  should preferably be addressed with a RFC1918 private address
#  scheme.
#
#  For this example, "eth0" is external and "eth1" is internal"
#
#  NOTE:  If this doesnt EXACTLY fit your configuration, you must
#         change the EXTIF or INTIF variables above. For example:
#
#            If you are a PPPoE or analog modem user:
#
#               EXTIF="ppp0"
#
EXTIF="eth0"
INTIF="eth1"
```

```
echo "  External Interface:  $EXTIF"
echo "  Internal Interface:  $INTIF"
echo "  ---"

# Specify your Static IP address here or let the script take care of it
# for you.
#
#   If you prefer to use STATIC addresses in your firewalls, un-# out the
#   static example below and # out the dynamic line.  If you don't care,
#   just leave this section alone.
#
#   If you have a DYNAMIC IP address, the ruleset already takes care of
#   this for you.  Please note that the different single and double quote
#   characters and the script MATTER.
#
#
#   PPP and DHCP (Cablemodem and DSL ) users:
#   ----------------------------------------
#   PPP: If you get your TCP/IP address via DHCP, **you will need ** to
#   enable the #   #ed out command below underneath the PPP section AND
#   replace the word "eth0" with the name of your EXTERNAL Internet
#   connection (ppp0, ippp0, etc) on the lines for "ppp-ip" and "extip".
#
#   DHCP and PPP users:  The remote DHCP or PPP server can and will change
#   IP addresses on you over time.  To deal with this, users should configure
#   their DHCP or PPP client to re-run the rc.firewall-* ruleset everytime
#   the IP address is changed.  Please see the "masq-and-dyn-addr" FAQ entry
#   in the IPMASQ howto for full details on how to do this.
#
#
# Determine the external IP automatically:
# ----------------------------------------
#
#  The following line will determine your external IP address.  This
#  line is somewhat complex and confusing but it will also work for
#  all NON-English Linux distributions:
#
EXTIP="`$IFCONFIG $EXTIF | $AWK \
 /$EXTIF/'{next}//{split($0,a,":");split(a[2],a," ");print a[1];exit}'`"


# For users who wish to use STATIC IP addresses:
#
#  # out the EXTIP line above and un-# out the EXTIP line below
#
#EXTIP="your.static.PPP.address"
echo "  External IP: $EXTIP"
echo "  ---"


# Assign the internal TCP/IP network and IP address
INTNET="192.168.0.0/24"
INTIP="192.168.0.1/32"
echo "  Internal Network: $INTNET"
echo "  Internal IP:      $INTIP"
```

```
echo "  ---"




# Setting a few other local variables
#
UNIVERSE="0.0.0.0/0"

#=========================================================================
#== No editing beyond this line is required for initial MASQ testing ==

# Need to verify that all modules have all required dependencies
#
echo "  - Verifying that all kernel modules are ok"
$DEPMOD -a

echo -en "    Loading kernel modules: "

# With the new IPTABLES code, the core MASQ functionality is now either
# modular or compiled into the kernel.  This HOWTO shows ALL IPTABLES
# options as MODULES.  If your kernel is compiled correctly, there is
# NO need to load the kernel modules manually.
#
#  NOTE: The following items are listed ONLY for informational reasons.
#        There is no reason to manual load these modules unless your
#        kernel is either mis-configured or you intentionally disabled
#        the kernel module autoloader.
#

# Upon the commands of starting up IP Masq on the server, the
# following kernel modules will be automatically loaded:
#
# NOTE:  Only load the IP MASQ modules you need.  All current IP MASQ
#        modules are shown below but are commented out from loading.
# ============================================================

#Load the main body of the IPTABLES module - "ip_tables"
#  - Loaded automatically when the "iptables" command is invoked
#
#  - Loaded manually to clean up kernel auto-loading timing issues
#
echo -en "ip_tables, "
#
#Verify the module isn't loaded.  If it is, skip it
#
if [ -z "` $LSMOD | $GREP ip_tables | $AWK {'print $1'} `" ]; then
   $MODPROBE ip_tables
fi



#Load the IPTABLES filtering module - "iptable_filter"
#
#  - Loaded automatically when filter policies are activated
```

```
#Load the stateful connection tracking framework - "ip_conntrack"
#
# The conntrack  module in itself does nothing without other specific
# conntrack modules being loaded afterwards such as the "ip_conntrack_ftp"
# module
#
#  - This module is loaded automatically when MASQ functionality is
#    enabled
#
#  - Loaded manually to clean up kernel auto-loading timing issues
#
echo -en "ip_conntrack, "
#
#Verify the module isn't loaded.  If it is, skip it
#
if [ -z "` $LSMOD | $GREP ip_conntrack | $AWK {'print $1'} `" ]; then
   $MODPROBE ip_conntrack
fi


#Load the FTP tracking mechanism for full FTP tracking
#
# Enabled by default -- insert a "#" on the next line to deactivate
#
echo -e "ip_conntrack_ftp, "
#
#Verify the module isn't loaded.  If it is, skip it
#
if [ -z "` $LSMOD | $GREP ip_conntrack_ftp | $AWK {'print $1'} `" ]; then
   $MODPROBE ip_conntrack_ftp
fi


#Load the IRC tracking mechanism for full IRC tracking
#
# Disabled by default -- insert a "#" on the next few lines to activate
#
# echo -en "                              ip_conntrack_irc, "
#
#Verify the module isn't loaded.  If it is, skip it
#
# if [ -z "` $LSMOD | $GREP ip_conntrack_irc | $AWK {'print $1'} `" ]; then
#    $MODPROBE ip_conntrack_irc
# fi


#Load the general IPTABLES NAT code - "iptable_nat"
#  - Loaded automatically when MASQ functionality is turned on
#
#  - Loaded manually to clean up kernel auto-loading timing issues
#
echo -en "iptable_nat, "
#
#Verify the module isn't loaded.  If it is, skip it
```

```
#
if [ -z "` $LSMOD | $GREP iptable_nat | $AWK {'print $1'} `" ]; then
   $MODPROBE iptable_nat
fi


#Loads the FTP NAT functionality into the core IPTABLES code
# Required to support non-PASV FTP.
#
# Enabled by default -- insert a "#" on the next line to deactivate
#
echo -e "ip_nat_ftp"
#
#Verify the module isn't loaded.  If it is, skip it
#
if [ -z "` $LSMOD | $GREP ip_nat_ftp | $AWK {'print $1'} `" ]; then
   $MODPROBE ip_nat_ftp
fi


#Loads the IRC NAT functionality (for DCC) into the core IPTABLES code
#
# DISABLED by default -- delete the "#" on the next few lines to activate
#
# echo -e "ip_nat_irc"
#
#Verify the module isn't loaded.  If it is, skip it
#
# if [ -z "` $LSMOD | $GREP ip_nat_irc | $AWK {'print $1'} `" ]; then
#    $MODPROBE ip_nat_irc
# fi


echo "  ---"

# Just to be complete, here is a partial list of some of the other
# IPTABLES kernel modules and their function.  Please note that most
# of these modules (the ipt ones) are automatically loaded by the
# master kernel module for proper operation and don't need to be
# manually loaded.
# ------------------------------------------------------------------
#
#    ip_nat_snmp_basic - this module allows for proper NATing of some
#                        SNMP traffic
#
#    iptable_mangle    - this target allows for packets to be
#                        manipulated for things like the TCPMSS
#                        option, etc.
#
# --
#
#    ipt_mark       - this target marks a given packet for future action.
#                     This automatically loads the ipt_MARK module
#
#    ipt_tcpmss     - this target allows to manipulate the TCP MSS
```

```
#                     option for braindead remote firewalls.
#                     This automatically loads the ipt_TCPMSS module
#
#    ipt_limit      - this target allows for packets to be limited to
#                     to many hits per sec/min/hr
#
#    ipt_multiport  - this match allows for targets within a range
#                     of port numbers vs. listing each port individually
#
#    ipt_state      - this match allows to catch packets with various
#                     IP and TCP flags set/unset
#
#    ipt_unclean    - this match allows to catch packets that have invalid
#                     IP/TCP flags set
#
#    iptable_filter - this module allows for packets to be DROPped,
#                     REJECTed, or LOGged.  This module automatically
#                     loads the following modules:
#
#                     ipt_LOG - this target allows for packets to be
#                               logged
#
#                     ipt_REJECT - this target DROPs the packet and returns
#                                  a configurable ICMP packet back to the
#                                  sender.


#CRITICAL:  Enable IP forwarding since it is disabled by default since
#
#           Redhat Users:  you may try changing the options in
#                          /etc/sysconfig/network from:
#
#                       FORWARD_IPV4=false
#                             to
#                       FORWARD_IPV4=true
#
echo "  Enabling forwarding.."
echo "1" > /proc/sys/net/ipv4/ip_forward


# Dynamic IP users:
#
#   If you get your IP address dynamically from SLIP, PPP, or DHCP,
#   enable the following option.  This enables dynamic-address hacking
#   which makes the life with Diald and similar programs much easier.
#
echo "  Enabling DynamicAddr.."
echo "1" > /proc/sys/net/ipv4/ip_dynaddr

echo "  ---"

#############################################################################
#
# Enable Stronger IP forwarding and Masquerading
#
```

```
#  NOTE:  In IPTABLES speak, IP Masquerading is a form of SourceNAT or SNAT.
#
#  NOTE #2:  The following is an example for an internal LAN address in the
#            192.168.0.x network with a 255.255.255.0 or a "24" bit subnet
#            mask connecting to the Internet on external interface "eth0".
#            This example will MASQ internal traffic out to the Internet
#            but not allow non-initiated traffic into your internal network.
#
#
#         ** Please change the above network numbers, subnet mask, and your
#

#Clearing any previous configuration
#
#  Unless specified, the defaults for INPUT, OUTPUT, and FORWARD to DROP
#
#    You CANNOT change this to REJECT as it isn't a vaild policy setting.
#    If you want REJECT, you must explictly REJECT at the end of a giving
#    INPUT, OUTPUT, or FORWARD chain
#
echo "  Clearing any existing rules and setting default policy to DROP.."
$IPTABLES -P INPUT DROP
$IPTABLES -F INPUT
$IPTABLES -P OUTPUT DROP
$IPTABLES -F OUTPUT
$IPTABLES -P FORWARD DROP
$IPTABLES -F FORWARD
$IPTABLES -F -t nat

#Not needed and it will only load the unneeded kernel module
#
#$IPTABLES -F -t mangle


# Delete all User-specified chains
$IPTABLES -X


# Reset all IPTABLES counters
$IPTABLES -Z


#Configuring specific CHAINS for later use in the ruleset
#
#  NOTE:  Some users prefer to have their firewall silently
#         "DROP" packets while others prefer to use "REJECT"
#         to send ICMP error messages back to the remote
#         machine.  The default is "REJECT" but feel free to
#         change this below.
#
# NOTE: Without the --log-level set to "info", every single
#       firewall hit will goto ALL vtys.  This is a very big
#       pain.
#
echo "  Creating a DROP chain.."
```

```
$IPTABLES -N reject-and-log-it
$IPTABLES -A reject-and-log-it -j LOG --log-level info
$IPTABLES -A reject-and-log-it -j REJECT

echo -e "\n   - Loading INPUT rulesets"


##############################################################################
# INPUT: Incoming traffic from various interfaces.  All rulesets are
#        already flushed and set to a default policy of DROP.
#

# loopback interfaces are valid.
#
$IPTABLES -A INPUT -i lo -s $UNIVERSE -d $UNIVERSE -j ACCEPT


# local interface, local machines, going anywhere is valid
#
$IPTABLES -A INPUT -i $INTIF -s $INTNET -d $UNIVERSE -j ACCEPT


# remote interface, claiming to be local machines, IP spoofing, get lost
#
$IPTABLES -A INPUT -i $EXTIF -s $INTNET -d $UNIVERSE -j reject-and-log-it


# external interface, from any source, for ICMP traffic is valid
#
#  If you would like your machine to "ping" from the Internet,
#  enable this next line
#
#$IPTABLES -A INPUT -i $EXTIF -p ICMP -s $UNIVERSE -d $EXTIP -j ACCEPT


# remote interface, any source, going to the MASQ servers IP address is valid
#
#  ENABLE this line if you want ALL Internet traffic to connect to your
#  the various servers running on the MASQ server.  This includes
#  web servers, ssh servers, dns servers, etc.
#
#  I DON'T recommend you enable this rule.  Instead, only enable specific
#  access to select server ports under the "OPTIONAL INPUT Section".
#  An example of enabling HTTP (WWW) has been given below:
#
#
#$IPTABLES -A INPUT -i $EXTIF -s $UNIVERSE -d $EXTIP -j ACCEPT


# Allow any related traffic coming back to the MASQ server in.
#
#  STATEFULLY TRACKED
#
$IPTABLES -A INPUT -i $EXTIF -s $UNIVERSE -d $EXTIP -m state --state \
 ESTABLISHED,RELATED -j ACCEPT
```

```
# ----- Begin OPTIONAL INPUT Section -----
#

# DHCPd - Enable the following lines if you run an INTERNAL DHCPd server
#
#$IPTABLES -A INPUT -i $INTIF -p tcp --sport 68 --dport 67 -j ACCEPT
#$IPTABLES -A INPUT -i $INTIF -p udp --sport 68 --dport 67 -j ACCEPT

# HTTPd - Enable the following lines if you run an EXTERNAL WWW server
#
#    NOTE:  This is NOT needed for simply enabling PORTFW.  This is ONLY
#           for users that plan on running Apache on the MASQ server itself
#
#echo -e "      - Allowing EXTERNAL access to the WWW server"
#$IPTABLES -A INPUT -i $EXTIF -m state --state NEW,ESTABLISHED,RELATED \
# -p tcp -s $UNIVERSE -d $EXTIP --dport 80 -j ACCEPT

#
# ----- End OPTIONAL INPUT Section -----


# Catch all rule, all other incoming is denied and logged.
#
$IPTABLES -A INPUT -s $UNIVERSE -d $UNIVERSE -j reject-and-log-it


# ---------------------------------------------------------------------

echo -e "   - Loading OUTPUT rulesets"

##############################################################################
# OUTPUT: Outgoing traffic from various interfaces.  All rulesets are
#         already flushed and set to a default policy of DROP.
#

# Workaround bug in netfilter
# See http://www.netfilter.org/security/2002-04-02-icmp-dnat.html
#
$IPTABLES -A OUTPUT -m state -p icmp --state INVALID -j DROP

# loopback interface is valid.
#
$IPTABLES -A OUTPUT -o lo -s $UNIVERSE -d $UNIVERSE -j ACCEPT


# local interfaces, any source going to local net is valid
#
$IPTABLES -A OUTPUT -o $INTIF -s $EXTIP -d $INTNET -j ACCEPT


# local interface, MASQ server source going to the local net is valid
#
$IPTABLES -A OUTPUT -o $INTIF -s $INTIP -d $INTNET -j ACCEPT
```

```
# outgoing to local net on remote interface, stuffed routing, deny
#
$IPTABLES -A OUTPUT -o $EXTIF -s $UNIVERSE -d $INTNET -j reject-and-log-it


# anything else outgoing on remote interface is valid
#
$IPTABLES -A OUTPUT -o $EXTIF -s $EXTIP -d $UNIVERSE -j ACCEPT


# ----- Begin OPTIONAL OUTPUT Section -----
#

# DHCPd - Enable the following lines if you run an INTERNAL DHCPd server
#         - Remove BOTH #s all the #s if you need this functionality.
#
#$IPTABLES -A OUTPUT -o $INTIF -p tcp -s $INTIP --sport 67 \
# -d 255.255.255.255 --dport 68 -j ACCEPT
#$IPTABLES -A OUTPUT -o $INTIF -p udp -s $INTIP --sport 67 \
# -d 255.255.255.255 --dport 68 -j ACCEPT

#
# ----- End OPTIONAL OUTPUT Section -----


# Catch all rule, all other outgoing is denied and logged.
#
$IPTABLES -A OUTPUT -s $UNIVERSE -d $UNIVERSE -j reject-and-log-it


echo -e "   - Loading FORWARD rulesets"

########################################################################
# FORWARD: Enable Forwarding and thus IPMASQ
#

# ----- Begin OPTIONAL FORWARD Section -----
#
#  Put PORTFW commands here
#
# ----- End OPTIONAL FORWARD Section -----


echo "     - FWD: Allow all connections OUT and only existing/related IN"
$IPTABLES -A FORWARD -i $EXTIF -o $INTIF -m state --state ESTABLISHED,RELATED \
 -j ACCEPT
$IPTABLES -A FORWARD -i $INTIF -o $EXTIF -j ACCEPT

# Catch all rule, all other forwarding is denied and logged.
#
$IPTABLES -A FORWARD -j reject-and-log-it
```

```
echo "     - NAT: Enabling SNAT (MASQUERADE) functionality on $EXTIF"
#
#More liberal form
#$IPTABLES -t nat -A POSTROUTING -o $EXTIF -j MASQUERADE
#
#Stricter form
$IPTABLES -t nat -A POSTROUTING -o $EXTIF -j SNAT --to $EXTIP


##########################################################################
echo -e "\nrc.firewall-iptables-stronger $FWVER done.\n"
```

<rc.firewall-iptables-stronger STOP>

To automatically start this stronger firewall ruleset at the proper time, please see the end of the Section 3.4.2 section for full details. Please make sure you make the correct "rc.firewall-iptables" to "rc.firewall-iptables-stronger" substitutions!!

6.4.2. Stronger IP Firewall (IPCHAINS) rulesets

This section provides a more in-depth guide to using the 2.2.x firewall tool, IPCHAINS. See above sections for IPFWADM rulesets.

This example is for a firewall/masquerade system behind a PPP link with a static PPP address (dynamic PPP instructions are included but disabled). The trusted interface is 192.168.0.1 and the PPP interface IP address has been changed to protect the guilty :-). I have listed each incoming and outgoing interface individually to catch IP spoofing as well as stuffed routing and/or masquerading. A nything not explicitly allowed is **FORBIDDEN** (well.. rejected actually). If your IP MASQ box breaks after implementing this rc.firewall-ipchains-stronger script, be sure that you edit it for your configuration and check your /var/log/messages or /var/adm/messages SYSLOG file for any firewall errors.

For more comprehensive examples of a strong IP Masqueraded IPFWADM rulesets for PPP, Cablemodem users, etc., please see TrinityOS - Section 10 and GreatCircle's Firewall WWW page

NOTE #1: --- UPDATE YOUR KERNEL --- Linux 2.2.x kernels less than version 2.2.20 contain several different security vulnerabilities (some were MASQ specific). Kernels less than 2.2.20 have a few local vulnerabilities. Kernel versions less than 2.2.16 have a TCP root exploit vulnerability and versions less than 2.2.11 have a IPCHAINS fragmentation bug. Because of these issues, users running a firewall with strong IPCHAINS rulesets are open to possible instrusion. Please upgrade your kernel to a fixed version.

NOTE #2: If you get a dynamically assigned TCP/IP address from your ISP (PPP, DSL, Cablemodems, etc.), you **CANNOT load** this strong ruleset upon booting. You will either need to reload this firewall ruleset EVERY TIME you get a new IP address or make your /etc/rc.d/rc.firewall-ipchains-stronger ruleset more intelligent. To do this for various types of connections such as PPP or DHCP users, please see the Section 7.8 FAQ entry for all the details.

Please also be aware that there are several GUI Firewall creation tools available as well. Please see Chapter 7for full details.

Lastly, if you are using a STATIC PPP IP address, change the "EXTIF="your.static.PPP.address"" line to reflect your address.

<rc.firewall-ipchains-stronger START>

```
#!/bin/sh
#
# /etc/rc.d/rc.firewall-ipchains-stronger: An example of a Stronger IPCHAINS
#                                          firewall ruleset for 2.2 kernels
#
FWVER=0.75s
#
# Log:
#  0.75s - Updated the commands for dynamically addresses machines and
#           to point to an expanded FAQ section for more information
#
#  0.74s - renamed from rc.firewall-2.2-stronger to
#          rc.firewall-ipchains-stronger to better reflect that this ruleset can
#          can run on different major kernel versions
#        - removed unused SED variable
#  0.73s - Added additional comments to make PORTFW configs more obvious
#  0.72s - #ed out the rule that would allow all traffic destined for the
#          MASQ server itself to be accepted.  Use the OPTIONAL INPUT
#          section to only allow explicit services.
#        - Fixed an INTLAN rule that was allowing traffic from ANY IP address
#          instead of the proper INTIP IP address only.  This aligns the
#          IPCHAINS ruleset with the IPTABLES and IPFWADM ruleset examples
#  0.71s - ruleset now uses modprobe instead of insmod
#  0.70s - Added missing execution variables
#        - fixed a missing -p tcp for the commented HTTPd section
#  0.65s - Added comments HTTPd rules to the INPUT and OUTPUT section
#        - Added a comment where to insert IPPORTFW commands
#  0.60s - Changed the EXTIP command to work on NON-English distros
#        - Updated the CASE of some of the script variables
#

echo -e "\nLoading rc.firewall-ipchains-stronger : version $FWVER..\n"


# The location of various iptables and other shell programs
#
#   If your Linux distribution came with a copy of iptables, most
#   likely it is located in /sbin.  If you manually compiled
#   iptables, the default location is in /usr/local/sbin
#
# ** Please use the "whereis iptables" command to figure out
# ** where your copy is and change the path below to reflect
# ** your setup
#
IPCHAINS=/sbin/ipchains
LSMOD=/sbin/lsmod
DEPMOD=/sbin/depmod
MODPROBE=/sbin/modprobe
GREP=/bin/grep
AWK=/bin/awk
IFCONFIG=/sbin/ifconfig

PATH=/sbin:/bin:/usr/sbin:/usr/bin
```

```
# Global variables
# ----------------

# ALL PPP and DHCP users must set this for the correct EXTERNAL and
#  INTERNAL interfaces names.  Examples:  eth0, ppp0, ippp0, etc.
#  See more info about this below.
#
EXTIF="ppp0"
INTIF="eth0"

# The INTERNAL IP address
#
INTIP="192.168.0.1/32"
INTNET="192.168.0.0/24"
echo "  Internal IP:      $INTIP"
echo "  Internal Network: $INTNET"



# Load all required IP MASQ modules
#
#   NOTE:  Only load the IP MASQ modules you need.  All current IP MASQ modules
#          are shown below but are commented from loading.

# Needed to initially load modules
#
$DEPMOD -a

# Supports the proper masquerading of FTP file transfers using the PORT method
#
$MODPROBE ip_masq_ftp

# Supports the masquerading of RealAudio over UDP.  Without this module,
#       RealAudio WILL function but in TCP mode.  This can cause a reduction
#       in sound quality
#
$MODPROBE ip_masq_raudio

# Supports the masquerading of IRC DCC file transfers
#
#$MODPROBE ip_masq_irc



# Supports the masquerading of Quake and QuakeWorld by default.  These modules are
#   for multiple users behind the Linux MASQ server.  If you are going to
#   play Quake I, II, and III, use the second example.
#
#   NOTE:  If you get ERRORs loading the QUAKE module, you are running an old
#   -----  kernel that has bugs in it.  Please upgrade to the newest kernel.
#
#Quake I / QuakeWorld (ports 26000 and 27000)
#$MODPROBE ip_masq_quake
#
#Quake I/II/III / QuakeWorld (ports 26000, 27000, 27910, 27960)
#$MODPROBE ip_masq_quake 26000,27000,27910,27960
```

```
# Supports the masquerading of the CuSeeme video conferencing software
#
#$MODPROBE ip_masq_cuseeme

#Supports the masquerading of the VDO-live video conferencing software
#
#$MODPROBE ip_masq_vdolive


#CRITICAL:  Enable IP forwarding since it is disabled by default
#
#           Redhat Users:  you may try changing the options in
#                          /etc/sysconfig/network from:
#
#                       FORWARD_IPV4=false
#                             to
#                       FORWARD_IPV4=true
#
echo "1" > /proc/sys/net/ipv4/ip_forward


#CRITICAL:  Enable automatic IP defragmentation since it is disabled by default
#           in 2.2.x kernels
#
#           This used as a compile-time option but the behavior was changed
#           in 2.2.12.  It should also be noted that some distributions have
#           removed this option from the /proc table.  If this entry isn't
#           present in your /proc, don't worry about it.
#
echo "1" > /proc/sys/net/ipv4/ip_always_defrag


# Dynamic IP users:
#
#   If you get your IP address dynamically from SLIP, PPP, or DHCP, enable this
#   following option.  This enables dynamic-ip address hacking in IP MASQ,
#   making life with Diald and similar programs much easier.
#
#echo "1" > /proc/sys/net/ipv4/ip_dynaddr


# Enable the LooseUDP patch which some Internet-based games require
#
#  If you are trying to get an Internet game to work through your IP MASQ box,
#  and you configured it to the best of your ability without it working, try
#  enabling this option (delete the "#" character).  This option is disabled
#  by default due to possible internal machine UDP port scanning
#  vulnerabilities.
#
#echo "1" > /proc/sys/net/ipv4/ip_masq_udp_dloose


# Specify your Static IP address here.
```

```
#
#   If you have a DYNAMIC IP address, you need to make this ruleset recognize
#   your IP address everytime you get a new IP.  To do this, enable the
#   following one-line script.  (Please note that the different single and
#   double quote characters MATTER).
#
#
#   DHCP users (Cablemodem and DSL ) users:
#   ---------------------------------------
#   If you get your TCP/IP address via DHCP, **you will need ** to enable the
#   #ed out command below underneath the PPP section AND replace the word
#   "ppp0" with the name of your EXTERNAL Internet connection (eth0, eth1, etc)
#   on the lines for "ppp-ip" and "EXTIP".
#
#   DHCP and PPP users:  The remote DHCP or PPP server can and will change
#   IP addresses on you over time.  To deal with this, users should configure
#   their DHCP or PPP client to re-run the rc.firewall-* ruleset everytime
#   the IP address is changed.  Please see the "masq-and-dyn-addr" FAQ entry
#   in the IPMASQ howto for full details on how to do this.
#
#
# Determine the external IP automatically:
# ----------------------------------------
#
#  The following line will determine your external IP address.  This
#  line is somewhat complex and confusing but it will also work for
#  all NON-English Linux distributions.
#
#   Make sure the EXTIF variable above is set to reflect the name
#   of your Internet connection
#
EXTIP="`$IFCONFIG $EXTIF | $AWK \
 /$EXTIF/'{next}//{split($0,a,":");split(a[2],a," ");print a[1];exit}'`"



# MASQ timeouts
#
#   2 hrs timeout for TCP session timeouts
#  10 sec timeout for traffic after the TCP/IP "FIN" packet is received
#  60 sec timeout for UDP traffic (MASQ'ed ICQ users must enable a 30sec
#     firewall timeout in ICQ itself)
#
$IPCHAINS -M -S 7200 10 60

################################################################################
# Incoming, flush and set default policy of reject. Actually the default policy
# is irrelevant because there is a catch all rule with deny and log.
#
$IPCHAINS -F input
$IPCHAINS -P input REJECT

# local interface, local machines, going anywhere is valid
#
$IPCHAINS -A input -i $INTIF -s $INTNET -d 0.0.0.0/0 -j ACCEPT
```

```
# remote interface, claiming to be local machines, IP spoofing, get lost
#
$IPCHAINS -A input -i $EXTIF -s $INTNET -d 0.0.0.0/0 -l -j REJECT


# remote interface, any source, going to the MASQ servers IP address is valid
#
#  ENABLE this line if you want ALL Internet traffic to connect to your
#  the various servers running on the MASQ server.  This includes
#  web servers, ssh servers, dns servers, etc.
#
#  I DON'T recommend you enable this rule.  Instead, only enable specific
#  access to select server ports under the "OPTIONAL INPUT Section".
#  An example of enabling HTTP (WWW) has been given below:
#
#
#$IPCHAINS -A input -i $EXTIF -s 0.0.0.0/0 -d $EXTIP/32 -j ACCEPT


# loopback interface is valid.
#
$IPCHAINS -A input -i lo -s 0.0.0.0/0 -d 0.0.0.0/0 -j ACCEPT


# ----- Begin OPTIONAL INPUT Section -----
#

# HTTPd - Enable the following lines if you either run a WWW server on
#         the IPMASQ server -OR- plan on PORTFW'ing HTTP traffic to
#         an internal WWW server
#
#$IPCHAINS -A input -i $EXTIF -p tcp -s 0.0.0.0/0 -d $EXTIP 80 -j ACCEPT

#
# ----- End OPTIONAL INPUT Section -----


# catch all rule, all other incoming is denied and logged. pity there is no
# log option on the policy but this does the job instead.
#
$IPCHAINS -A input -s 0.0.0.0/0 -d 0.0.0.0/0 -l -j REJECT

################################################################################
# Outgoing, flush and set default policy of reject. Actually the default policy
# is irrelevant because there is a catch all rule with deny and log.
#
$IPCHAINS -F output
$IPCHAINS -P output REJECT

# local interface, MASQ server source going to the local net is valid
#
$IPCHAINS -A output -i $INTIF -s $INTIP -d $INTNET -j ACCEPT

# outgoing to local net on remote interface, stuffed routing, deny
```

```
#
$IPCHAINS -A output -i $EXTIF -s 0.0.0.0/0 -d $INTNET -l -j REJECT

# outgoing from local net on remote interface, stuffed masquerading, deny
#
$IPCHAINS -A output -i $EXTIF -s $INTNET -d 0.0.0.0/0 -l -j REJECT

# anything else outgoing on remote interface is valid
#
$IPCHAINS -A output -i $EXTIF -s $EXTIP/32 -d 0.0.0.0/0 -j ACCEPT

# loopback interface is valid.
#
$IPCHAINS -A output -i lo -s 0.0.0.0/0 -d 0.0.0.0/0 -j ACCEPT


# ----- Begin OPTIONAL OUTPUT Section -----
#

# HTTPd - Enable the following lines if you either run a WWW server on
#         the IPMASQ server -OR- plan on PORTFW'ing HTTP traffic to
#         an internal WWW server
#
#$IPCHAINS -A output -i $EXTIF -p tcp -s $EXTIP 80 -d 0.0.0.0/0 -j ACCEPT

#
# ----- End OPTIONAL OUTPUT Section -----

# catch all rule, all other outgoing is denied and logged. pity there is no
# log option on the policy but this does the job instead.
#
$IPCHAINS -A output -s 0.0.0.0/0 -d 0.0.0.0/0 -l -j REJECT

################################################################################
# Forwarding, flush and set default policy of deny. Actually the default policy
# is irrelevant because there is a catch all rule with deny and log.
#
$IPCHAINS -F forward
$IPCHAINS -P forward DENY


# ----- Begin OPTIONAL FORWARD Section -----
#
#   Put PORTFW commands here
#
# ----- End OPTIONAL FORWARD Section -----


# Masquerade from local net on local interface to anywhere.
#
$IPCHAINS -A forward -i $EXTIF -s $INTNET -d 0.0.0.0/0 -j MASQ
#
# catch all rule, all other forwarding is denied and logged. pity there is no
# log option on the policy but this does the job instead.
#
```

```
$IPCHAINS -A forward -s 0.0.0.0/0 -d 0.0.0.0/0 -l -j REJECT

#End of file.
```

<rc.firewall-ipchains-stronger STOP>

To automatically start this stronger firewall ruleset at the proper time, please see the end of the Section 3.4.2 section for full details. Please make sure you make the correct "rc.firewall-ipchains" to "rc.firewall-ipchains-stronger" substitutions!!

With IPCHAINS, you can block traffic to a particular site using the "input", "output", and/or "forward" rules. Remember that the set of rules are scanned from top to bottom and "-A" tells IPCHIANS to "append" this new rule to the existing set of rules. So with this in mind, any specific restrictions need to come before any global rules. For example:

Using "input" rules:

Probably the fastest and most efficient method to block traffic, but this method only stops the MASQed machines and NOT the firewall machine itself. Of course, you might want to allow that combination.

Anyway, to block 204.50.10.13:

```
In the /etc/rc.d/rc.firewall-ipchains-stronger ruleset:
... start of "input" rules ...

# reject and log local interface, local machines going to 204.50.10.13
#
ipchains -A input -s 192.168.0.0/24 -d 204.50.10.13/32 -l -j REJECT


# local interface, local machines, going anywhere is valid
#
ipchains -A input -s 192.168.0.0/24 -d 0.0.0.0/0 -l -j ACCEPT


... end of "input" rules ...
```

Using "output" rules:

This is the slower method to block traffic because the packets must go through masquerading before they are dropped. Yet, this rule even stops the firewall machine from accessing the forbidden site.

... start of "output" rules ... # reject and log outgoing to 204.50.10.13 # ipchains -A output -s $ppp_ip/32 -d 204.50.10.13/32 -l -j REJECT # anything else outgoing on remote interface is valid # ipchains -A output -s $ppp_ip/32 -d 0.0.0.0/0 -l -j ACCEPT ... end of "output" rules ...

Using "forward" rules:

Probably slower than "input" rules for blocking traffic, this only stops masqueraded machines (e.g. internal machines). The firewall machine can still reach forbidden site(s).

... start of "forward" rules ... # Reject and log from local net on PPP interface to 204.50.10.13. # ipchains -A forward -i ppp0 -s 192.168.0.0/24 -d 204.50.10.13/32 -l -j REJECT # Masquerade from local net on local interface to anywhere. # ipchains -A forward -

i ppp0 -s 192.168.0.0/24 -d 0.0.0.0/0 -j MASQ ... end of "forward" rules ...

No need for a special rule to allow machines on the 192.168.0.0/24 network to go to 204.50.11.0. Why? It is already covered by the global MASQ rule.

NOTE: Unlike IPFWADM, IPCHIANS has only one way of coding the interfaces name. IPCHAINS uses the "-i eth0" option where as IPFWADM had both "-W" for the interface name and "-V" for the interface's IP address.

6.4.3. Stronger IP Firewall (IPFWADM) Rulesets

This section provides a more in-depth guide on using the 2.0.x firewall tool, IPFWADM. See below for IPCHAINS rulesets

This example is for a firewall/masquerade system behind a PPP link with a static PPP address (dynamic PPP instructions are included but disabled). The trusted interface is 192.168.0.1 and the PPP interface IP address has been changed to protect the guilty :). I have listed each incoming and outgoing interface individually to catch IP spoofing as well as stuffed routing and/or masquerading. Anything not explicitly allowed is **FORBIDDEN** (well.. rejected, actually). If your IP MASQ box breaks after implementing this rc.firewall-ipfwadm-stronger script, be sure that you edit it for your configuration and check your /var/log/messages or /var/adm/messages SYSLOG file for any firewall errors.

For more comprehensive examples of a strong IP Masqueraded IPFWADM rulesets for PPP, Cablemodem users, etc., please see TrinityOS - Section 10 and GreatCircle's Firewall WWW page

NOTE #2: If you get a dynamically assigned TCP/IP address from your ISP (PPP, DSL, Cablemodems, etc.), you **CANNOT load** this strong ruleset upon booting. You will either need to reload this firewall ruleset EVERY TIME you get a new IP address or make your /etc/rc.d/rc.firewall-ipchains-stronger ruleset more intelligent. To do this for various types of connections such as PPP or DHCP users, please see the Section 7.8 FAQ entry for all the details.

Please also be aware that there are several GUI Firewall creation tools available as well. Please see Chapter 7for full details.

Lastly, if you are using a STATIC PPP IP address, change the "ppp_ip="your.static.PPP.address"" line to reflect your address.

<rc.firewall-ipfwadm-stronger START>

```
#!/bin/sh
#
# /etc/rc.d/rc.firewall-ipfwadm-stronger: An example of a semi-STRONG
#                                         IPFWADM firewall ruleset for 2.0 kernels
#
FWVER=0.74s
#
# Log:
#  0.74s - Updated the commands for dynamically addresses machines and
#           to point to an expanded FAQ section for more information
#
#  0.73s - renamed from rc.firewall-2.0-stronger to
#          rc.firewall-ipfwadm-stronger
#
#  0.72s - #ed out the rule that would allow all traffic destined for the
```

```
#          MASQ server itself to be accepted.  Use the OPTIONAL INPUT
#          section to only allow explicit services.


PATH=/sbin:/bin:/usr/sbin:/usr/bin

# testing, wait a bit then clear all firewall rules.
# uncomment the following lines if you want the firewall to automatically
# disable after 10 minutes.
#
# Disabled by default
#
# (sleep 600; \
# ipfwadm -I -f; \
# ipfwadm -I -p accept; \
# ipfwadm -O -f; \
# ipfwadm -O -p accept; \
# ipfwadm -F -f; \
# ipfwadm -F -p accept; \
# ) &


# Load all required IP MASQ modules
#
#   NOTE:  Only load the IP MASQ modules you need.  All current IP MASQ modules
#          are shown below but are commented from loading.

# Needed to initially load modules
#
/sbin/depmod -a

# Supports the proper masquerading of FTP file transfers using the PORT method
#
/sbin/modprobe ip_masq_ftp

# Supports the masquerading of RealAudio over UDP.  Without this module,
#       RealAudio WILL function but in TCP mode.  This can cause a reduction
#       in sound quality
#
#/sbin/modprobe ip_masq_raudio

# Supports the masquerading of IRC DCC file transfers
#
#/sbin/modprobe ip_masq_irc


# Supports the masquerading of Quake and QuakeWorld by default.  This modules is
#   for multiple users behind the Linux MASQ server.  If you are going to
#   play Quake I, II, and III, use the second example.
#
#   NOTE:  If you get ERRORs loading the QUAKE module, you are running an old
#   -----  kernel that has bugs in it.  Please upgrade to the newest kernel.
#
#Quake I / QuakeWorld (ports 26000 and 27000)
#/sbin/modprobe ip_masq_quake
```

```
#
#Quake I/II/III / QuakeWorld (ports 26000, 27000, 27910, 27960)
#/sbin/modprobe ip_masq_quake 26000,27000,27910,27960


# Supports the masquerading of the CuSeeme video conferencing software
#
#/sbin/modprobe ip_masq_cuseeme

#Supports the masquerading of the VDO-live video conferencing software
#
#/sbin/modprobe ip_masq_vdolive


#CRITICAL:  Enable IP forwarding, since it is disabled by default
#
#           Redhat Users:  you may try changing the options in /etc/sysconfig/
network
#                          from:
#
#                       FORWARD_IPV4=false
#                             to
#                       FORWARD_IPV4=true
#
echo "1" > /proc/sys/net/ipv4/ip_forward


#CRITICAL:  Enable automatic IP defragmenting since it is disabled by default
#           in 2.2.x kernels
#
#           This used to be a compile-time option but the behavior was changed
#           in 2.2.12
#
echo "1" > /proc/sys/net/ipv4/ip_always_defrag


# Dynamic IP users:
#
#   If you get your IP address dynamically from SLIP, PPP, or DHCP, enable this
#   following option.  This allows dynamic-ip address hacking in IP MASQ,
#   making the life with Diald and similar programs much easier.
#
#echo "1" > /proc/sys/net/ipv4/ip_dynaddr


# Specify your Static IP address here.
#
#   If you have a DYNAMIC IP address, you need to make this ruleset understand
#   your IP address everytime you get a new IP.  To do this, enable the
#   following one-line script.  (Please note that the different single and
#   double quote characters MATTER).
#
#
#   DHCP (Cablemodem and DSL) and PPP users:
#   ----------------------------------------
```

```
#   If you get your TCP/IP address a dynamic IP address **you will need ** to
#   enable the #ed out command below underneath the PPP section AND replace the
word
#   "ppp0" with the name of your EXTERNAL Internet connection (eth0, eth1,
#   etc).
#
#   DHCP and PPP users:  The remote DHCP or PPP server can and will change
#   IP addresses on you over time.  To deal with this, users should configure
#   their DHCP or PPP client to re-run the rc.firewall-* ruleset everytime
#   the IP address is changed.  Please see the "masq-and-dyn-addr" FAQ entry
#   in the IPMASQ howto for full details on how to do this.
#
#
# PPP and DHCP Users:
# -------------------
# Remove the # on the line below and place a # in front of the line after that.
#
#ppp_ip="`/sbin/ifconfig ppp0 | grep 'inet addr' | awk '{print $2}' | sed -e 's/.
*://'`"
#
ppp_ip="your.static.PPP.address"


# MASQ timeouts
#
#   2 hrs timeout for TCP session timeouts
#  10 sec timeout for traffic after the TCP/IP "FIN" packet is received
#  60 sec timeout for UDP traffic (MASQ'ed ICQ users must enable a 30sec
#     firewall timeout in ICQ itself)
#
/sbin/ipfwadm -M -s 7200 10 60


################################################################################
# Incoming, flush and set default policy of reject. Actually the default policy
# is irrelevant because there is a catch all rule with deny and log.
#
/sbin/ipfwadm -I -f
/sbin/ipfwadm -I -p reject

# local interface, local machines, going anywhere is valid
#
/sbin/ipfwadm -I -a accept -V 192.168.0.1 -S 192.168.0.0/24 -D 0.0.0.0/0

# remote interface, claiming to be local machines, IP spoofing, get lost
#
/sbin/ipfwadm -I -a reject -V $ppp_ip -S 192.168.0.0/24 -D 0.0.0.0/0 -o


# remote interface, any source, going to the MASQ servers IP address is valid
#
#  ENABLE this line if you want ALL Internet traffic to connect to your
#  the various servers running on the MASQ server.  This includes
#  web servers, ssh servers, dns servers, etc.
#
```

```
#  I DON'T recommend you enable this rule.  Instead, only enable specific
#  access to select server ports under the "OPTIONAL INPUT Section".
#  An example of enabling HTTP (WWW) has been given below:
#
#
#/sbin/ipfwadm -I -a accept -V $ppp_ip -S 0.0.0.0/0 -D $ppp_ip/32


# loopback interface is valid.
#
/sbin/ipfwadm -I -a accept -V 127.0.0.1 -S 0.0.0.0/0 -D 0.0.0.0/0


# catch all rule, all other incoming is denied and logged. pity there is no
# log option on the policy but this does the job instead.
#
/sbin/ipfwadm -I -a reject -S 0.0.0.0/0 -D 0.0.0.0/0 -o


################################################################################
# Outgoing, flush and set default policy of reject. Actually the default policy
# is irrelevant because there is a catch all rule with deny and log.
#
/sbin/ipfwadm -O -f
/sbin/ipfwadm -O -p reject

# local interface, MASQ server source going to the local net is valid
#
/sbin/ipfwadm -O -a accept -V 192.168.0.1 -S 0.0.0.0/0 -D 192.168.0.0/24

# outgoing to local net on remote interface, stuffed routing, deny
#
/sbin/ipfwadm -O -a reject -V $ppp_ip -S 0.0.0.0/0 -D 192.168.0.0/24 -o

# outgoing from local net on remote interface, stuffed masquerading, deny
#
/sbin/ipfwadm -O -a reject -V $ppp_ip -S 192.168.0.0/24 -D 0.0.0.0/0 -o

# outgoing from local net on remote interface, stuffed masquerading, deny
#
/sbin/ipfwadm -O -a reject -V $ppp_ip -S 0.0.0.0/0 -D 192.168.0.0/24 -o

# anything else outgoing on remote interface is valid
#
/sbin/ipfwadm -O -a accept -V $ppp_ip -S $ppp_ip/32 -D 0.0.0.0/0

# loopback interface is valid.
#
/sbin/ipfwadm -O -a accept -V 127.0.0.1 -S 0.0.0.0/0 -D 0.0.0.0/0

# catch all rule, all other outgoing is denied and logged. pity there is no
# log option on the policy but this does the job instead.
#
/sbin/ipfwadm -O -a reject -S 0.0.0.0/0 -D 0.0.0.0/0 -o
```

```
#############################################################################
# Forwarding, flush and set default policy of deny. Actually the default policy
# is irrelevant because there is a catch all rule with deny and log.
#
/sbin/ipfwadm -F -f
/sbin/ipfwadm -F -p reject

# Masquerade from local net on local interface to anywhere.
#
/sbin/ipfwadm -F -a masquerade -W ppp0 -S 192.168.0.0/24 -D 0.0.0.0/0
#
# catch all rule, all other forwarding is denied and logged.  Pity there is no
# log option on the policy but this does the job instead.
#
/sbin/ipfwadm -F -a reject -S 0.0.0.0/0 -D 0.0.0.0/0 -o

#End of file.
```

<rc.firewall-ipfwadm-stronger STOP>

To automatically start this stronger firewall ruleset at the proper time, please see the end of the Section 3.4.3 section for full details. Please make sure you make the correct "rc.firewall-ipfwadm" to "rc.firewall-ipfwadm-stronger" substitutions!!

With IPFWADM, you can block traffic to a particular site using the -I, -O or -F rules. Remember that the set of rules are scanned top to bottom and "-a" tells IPFWADM to "append" this new rule to the existing set of rules. So with this in mind, any specific restrictions need to come before global rules. For example:

Using -I (input) rules:

Probably the fastest and most efficient method to block traffic but it only stops the MASQed machines, and NOT the the firewall machine itself. Of course, you might want to allow that combination.

Anyway, to block 204.50.10.13:

In the /etc/rc.d/rc.firewall-ipfwadm-stronger ruleset: ... start of -I rules ... # reject and log local interface, local machines going to 204.50.10.13 # /sbin/ipfwadm -I -a reject -V 192.168.0.1 -S 192.168.0.0/24 -D 204.50.10.13/32 -o # local interface, local machines, going anywhere is valid # /sbin/ipfwadm -I -a accept -V 192.168.0.1 -S 192.168.0.0/24 -D 0.0.0.0/0 ... end of -I rules ...

Using -O (output) rules:

This is the slower method to block traffic because the packets go through masquerading first before they are dropped. Yet, this rule even stops the firewall machine from accessing the forbidden site.

... start of -O rules ... # reject and log outgoing to 204.50.10.13 # /sbin/ipfwadm -O -a reject -V $ppp_ip -S $ppp_ip/32 -D 204.50.10.13/32 -o # anything else outgoing on remote interface is valid # /sbin/ipfwadm -O -a accept -V $ppp_ip -S $ppp_ip/32 -D 0.0.0.0/0 ... end of -O rules ...

Using -F (forward) rules:

Probably slower than -I (input) rules for blocking traffic, this still only stops masqueraded machines (e.g. internal machines). The firewall machine can still reach forbidden site(s).

... start of -F rules ... # Reject and log from local net on PPP interface to 204.50.10.13. # /sbin/ipfwadm -F -a reject -W ppp0 -S 192.168.0.0/24 -D 204.50.10.13/32 -o # Masquerade from local net on local interface to anywhere. # /sbin/ipfwadm -F -a masquerade -W ppp0 -S 192.168.0.0/24 -D 0.0.0.0/0 ... end of -F rules ...

There is no need for a special rule to allow machines on the 192.168.0.0/24 network to go to 204.50.11.0. Why? It is already covered by the global MASQ rule.

NOTE: There is more than one way of coding the interfaces in the above rules. For example instead of "-V 192.168.255.1" you can code "-W eth0", instead of "-V $ppp_ip" , you can use "-W ppp0". The "-V" method was phased out with the imgration to IPCHAINS, but for IPFWADM users, its more of a personal choice and documentation.

6.5. IP Masquerading multiple internal networks

Masquerading more than one internal network is fairly simple. You need to first make sure that all of your networks are running correctly (both internal and external). You then need to enable traffic to pass to both the other internal interfaces and to be MASQed to the Internet.

Next, you need to enable Masquerading on the INTERNAL interfaces. This example uses a total of THREE interfaces: EXTIF stands for the eth0 interface which is the EXTERNAL connection to the Internet. INTIF stands for the eth1 interface and is the 192.168.0.0 network. Finally, INTIF2 stands for the eth2 interface and is the 192.168.1.0 network. Both INTIF and INTIF2 will be MASQed out of interface eth0 or EXTIF. In your rc.firewall-* ruleset next to the existing MASQ at the very end of the ruleset, add the following:

6.5.1. iptables support for multiple internal lans

-

```
# 2.6.x and 2.4.x kernels with IPTABLES
#
# The following rules build upon the rc.firewall-iptables-stronger ruleset.
# Please see that ruleset in Section 6 for how all variables get set, etc.


  #Enable internal interfaces to communication between each other
  #
  $IPTABLES -A FORWARD -i $EXTIF -o $INTIF2 -m state --state ESTABLISHED,RELATED \
 -j ACCEPT
  $IPTABLES -A FORWARD -i $INTIF -o $INTIF2 -m state --state ESTABLISHED,RELATED \
 -j ACCEPT
  $IPTABLES -A FORWARD -i $INTIF2 -o $INTIF -m state --state ESTABLISHED,RELATED \
 -j ACCEPT

  $IPTABLES -t nat -A POSTROUTING -o $EXTIF -j SNAT --to $EXTIP
```

6.5.2. ipchains support for multiple internal lans

```
# 2.2.x kernels with IPCHAINS
#
# The following rules build upon the rc.firewall-ipchains-stronger ruleset.
# Please see that ruleset in Section 6 for how all variables get set, etc.

  #Enable internal interfaces to communication between each other
  $IPCHAINS -A forward -i eth1 -d 192.168.0.0/24 -j ACCEPT
  $IPCHAINS -A forward -i eth2 -d 192.168.1.0/24 -j ACCEPT

  #Enable internal interfaces to MASQ out to the Internet
  $IPCHAINS -A forward -j MASQ -i eth0 -s 192.168.0.0/24 -d 0.0.0.0/0
  $IPCHAINS -A forward -j MASQ -i eth0 -s 192.168.1.0/24 -d 0.0.0.0/0
```

6.5.3. ipfwadm support for multiple internal lans

```
# 2.0.x kernels with IPFWADM
#
# The following rules build upon the rc.firewall-ipfwadm-stronger ruleset.
# Please see that ruleset in Section 6 for how all variables get set, etc.

  #Enable internal interfaces to communication between each other
  /sbin/ipfwadm -F -a accept -V 192.168.0.1 -D 192.168.1.0/24
  /sbin/ipfwadm -F -a accept -V 192.168.1.1 -D 192.168.0.0/24

  #Enable internal interfaces to MASQ out to the Internet
  /sbin/ipfwadm -F -a masq -W eth0 -S 192.168.0.0/24 -D 0.0.0.0/0
  /sbin/ipfwadm -F -a masq -W eth0 -S 192.168.1.0/24 -D 0.0.0.0/0
```

Please note that it is CORRECT to have "eth0" specified multiple times for the exmples shown above. The reason for this is the Linux kernel needs to know which interface is used for OUTGOING traffic. Since eth0 in the above examples is the Internet connection, it is listed for each internal interface.

6.6. IP Masquerade and Dial-on-Demand Connections

1. If you would like to setup your network to automatically dial up the Internet, either the **Diald** demand dial-up or new versions of the **PPPd** packages will be of great utility. Both Diald and PPPd are very powerful in their configuration flexibility.
2. To setup PPPd for Dial-on-Demand, pease check out the PPPd Homepage

3. To setup Diald, please check out the Diald Homepage or TrinityOS - Section 23
4. Once Dial on Demand and IP Masq have been setup properly, any MASQed client machines that initiate a web, telnet or ftp session will make the Linux box dynamically bring up its Internet link.
5. There is a timeout that will occur with the first connection. This is inevitable if you are using analog modems. The time taken to establish the modem link and the PPP connections may cause your client program (WWW browser, etc.) to stop. This isn't common though. If this does happen, just retry that Internet traffic request (say a WWW page) again and it should come up fine. You can also try setting **echo "1" > /proc/sys/net/ipv4/ip_dynaddr** kernel option to help with this initial setup.

6.7. Port Forwarding with IPTABLES or external tools like IPPORTFW, IPMASQADM, IPAUTOFW, REDIR, UDPRED, and other Port Forwarding tools

IPTABLES as well as IPPORTFW, IPAUTOFW, REDIR, UDPRED, and other programs offer generic TCP and/or UDP port forwarding for Linux IP Masquerade. These tools are typically used with or as a replacement for specific IP MASQ modules to get a specific network traffic through the MASQ server.

With port forwarders, you can redirect data connections from the Internet to an internal, privately addressed machine behind your IP MASQ server. This forwarding ability includes network protocols such as TELNET, WWW, and SMTP. Protocols such as FTP, legacy ICQ, and others require special handling via kernel modules (see below).

NOTE: If you are just looking to do simple port forwarding but you don't need Masquerading support, you don't have any choice. You will **STILL NEED** to enable IP Masquerading support in the kernel AND either run a IPTABLES, IPCHAINS, or IPFWADM ruleset to be able to use Linux's port forwarding tools.

So why all the different choices? IPTABLES, IPPORTFW, MARK (MFW), IPMASQADM (PORTFW or AUTOFW), IPAUTOFW, REDIR, and UDPRED (all URLs are in Section 3.2.3) are the various tools available to IP MASQ users to allow this type of functionality depending on their kernel version. Later, as the Linux IP Masquerade feature matured, many of these tools were eventually replaced by the IPTABLES or PORTFW and MARK systems which are far more intelligent solutions.

For the later 2.2.x kernels, the IPMASQADM tool combined the legacy IPAUTOFW and IPPORTFW 2.0.x kernel tools into one binary. Both the IPMASQADM tool for 2.2.x kernels as well as IPTABLES for the 2.6.x and 2.4.x kernels also supports a new mechanism called "MARK" or MFW for PORTFW functionality. The MARK system works where a specific IPTABLES or IPCHAINS ruleset would match a given packet sequence. Once matched, the tool would "mark" these packets. Later, the IPMASQADM tool or a specific IPTABLE "table" could be instructed to change these packets as needed and forward them off to their desired internal destination. Currently, this HOWTO doesn't cover the MARK solution but it will in the future.

Anyway, because of the availablity of the newer tools, it is *HIGHLY DISCOURAGED* to use the old tools such as IPAUTOFW (even AUTOFW in IPMASQADM) and REDIR because they don't properly notify the Linux kernel of their presence and can ultimately **CRASH** your Linux server with extreme use.

NOTE #2: With enabling PORTFW functionality in ANY 2.2/2.0 Linux kernel (2.6.x. and 2.4.x users won't use these specific tools anyway), **internal machines typically CANNOT use the same "external" PORTFWed IP address to access a given internal" machine.** To put it another way, PORTFW was only intended to be used with "external" computers on the Internet. If this is an issue for you, you can also use the REDIR tool for older 2.2.x and 2.0.x kernels to let internal machines get redirected to the internal servers too. 2.6.x and 2.4.x kernels users running IPTABLES solves this issue once and for all and is fully covered in a FAQ entry in Section 7.19 below. If you would like a technical explination on why this internal/external forwarding doesn't work, please page down towards the bottom of the 2.2.x PORTFW section for a note from Juan Jose Ciarlante.

NOTE #3: The forwarding of non-NAT friendly traffic such as FTP server traffic to an internal MASQed FTP server, known as

PORTFW FTP, is now fully supported in the 2.6.x and 2.4.x kernels as well as in the 2.2.x kernels via a BETA version FTP kernel module (does NOT come with the stock Linus kernels). It should also be noted that you can also PORTFW FTP traffic using an external FTP proxy program (not covered in this HOWTO). It should be noted that the Beta 2.2.x FTP kernel module code is still experimental and some people get better results simply using ACTIVE FTP sessions compared to PASSIVE connections. Interestingly enough, other people have seen the exact opposite behavior. Please let us know what your results are like. More about this is covered below in both the 2.2.x and 2.0.x sections as the solutions require the use of different patches.

WARNING! Before jumping right into installing ANY of these tools, it needs to be mentioned that network security can be an issue with *ANY* PORT FORWARD tool. The reason for this is because these tools basically create a hole in strong packet firewalls for the required TCP/UDP ports. Though this doesn't pose any threat to your Linux machine, it might be an issue to the PORTFW'ed internal machine(s). No worries though, this is what Steven Clarke (the author of IPPORTFW) had to say about that:

```
   "Port Forwarding is only called within masquerading functions so it
   fits inside the same IPFWADM/IPCHAINS rules. Masquerading is an extension
to
   IP forwarding. Therefore, ipportfw only sees a packet if it fits
   both the input and masquerading ipfwadm rule sets."
```

What that means in English is that if you have a strong packet firewall running, PORTFW doesn't directly bypass any of that security. You will still be able to allow or deny specific IPs and/or domains to this new PORTFW'ed resource if you so wish.

With this said, it's important to have a strong firewall ruleset. Please see Section 6.4.1, Section 6.4.2, and Section 6.4.3 for more details on getting strong rulesets.

- 2.6.x and 2.4.x kernels have PORTFW functional already built in using the IPTABLES tool. 2.2.x and 2.0.x kernel kernel users will need to re-compile the Linux kernel to support PORTFW. It should be noted that some Linux distribution kernels might have this already done for you.
- The latest 2.2.x kernel users will already have the PORTFW kernel option available to them though you might still need to recompile the kernel via the normal kernel "make" procedures.
- 2.0.x users will need to apply a simple kernel option patch to have access to then enable this via the normal kernel "make" procedures.

6.7.1. IPTABLES-based PORTFWD'ing: Using IPTABLES's PREROUTING option for 2.6.x and 2.4.x kernels

As mentioned before, a port forward or "PORTFW" allows a single or range of TCP/IP ports from the external side of the network to be forwarded into the inside network.

Unlike ALL previous Linux kernels, the 2.6.x and 2.4.x series kernels now allows for full PORTFW, PORTFW FTP, and PORTFW REDIR functionality within the "iptables" tool itself.

NOTE: Once you enable a port forwarder on say port 80 (forward WWW traffic through the MASQ server to an internal WWW server), that port will no longer be used by the Linux IP Masquerade server itself. To be more specific, if you have a WWW server already running on the MASQ server, enabling PORTFW will now give all Internet users acces to the WWW pages from the -INTERNAL- WWW server and not the pages on your IP MASQ server.

To enable port forwarding on an IPTABLES (2.6.x or 2.4.x kernel):

- Edit the /etc/rc.d/rc.firewall-iptables ruleset and place the lines shown below just ABOVE the "`FWD: Allow all`

`connections OUT and only existing and related ones IN`" line (in the "Optional FORWARD section"). Please **be sure** to replace the word "$EXTIP" with your specific Internet IP address.

- NOTE: Unlike the 2.2.x and 2.0.x kernels, PORTFWed traffic does *not* traverse the INPUT or OUTPUT rules. It only traverses the FORWARD rule.

- **NOTE:** If you get a dynamically assigned TCP/IP address from your ISP (PPP, DSL, Cablemodems, etc.), you **CANNOT load** this strong ruleset upon booting. You will either need to reload this firewall ruleset EVERY TIME you get a new IP address or make your /etc/rc.d/rc.firewall-ipchains-stronger ruleset more intelligent. To do this for various types of connections such as PPP or DHCP users, please see the Section 7.8 FAQ entry for all the details.

/etc/rc.d/rc.firewall-*

```
#echo "Enabling PORTFW Redirection on the external LAN.."
#
#   This will forward ALL port 80 traffic from the external IP address
#   to port 80 on the 192.168.0.10 machine
#
#   Be SURE that when you add these new rules to your rc.firewall-*, you
#   add them before a direct or implict DROP or REJECT.
#
PORTFWIP="192.168.0.10"


# NOTE:  If you are using the basic rc.firewall-iptables ruleset, you
#        will need to enable the following EXTIP option.  Users of the
#        rc.firewall-iptables-stronger ruleset already have this defined.
#
#  *PLEASE* look over the rc.firewall-iptables-stronger ruleset for more
#           specific issues regarding dynamic vs. static IP addresses
#
#
# Determine the external IP automatically:
# ----------------------------------------
#
#  The following line will determine your external IP address.  This
#  line is somewhat complex and confusing but it will also work for
#  all NON-English Linux distributions:
#
# DISABLED by default -- to enable, REMOVE both the "#" characters below
#
#EXTIP="`$IFCONFIG $EXTIF | $AWK \
#/$EXTIF/'{next}//{split($0,a,":");split(a[2],a," ");print a[1];exit}'`"


# Allow forwarding of new and existing port 80 connections from the external
# interface.  This rule is required as our default FORWARD policy is DENY.
#
$IPTABLES -A FORWARD -i $EXTIF -o $INTIF -p tcp --dport 80 -m state \
 --state NEW,ESTABLISHED,RELATED -j ACCEPT


#Enable PORTFW of this port 80 traffic from the external interface
#
$IPTABLES -A PREROUTING -t nat -p tcp -d $EXTIP --dport 80 -m state \
```

```
 --state NEW,ESTABLISHED,RELATED -j DNAT --to $PORTFWIP:80
```

That's it! Just re-run your /etc/rc.d/rc.firewall-iptables ruleset and test it out! If you would like to learn more about this, please see the Section 7.19

Running the rc.firewall-iptables-stronger ruleset? Good for you! To get PORTFW running with this ruleset, it's very easy. The following example is for HTTP (WWW) traffic to be PORTFWed to the IP address indicated by the $PORTFWIP variable:

- Take the ruleset lines shown above (for the generic rc.firewall-* ruleset) and place them just *after* the "`#FORWARD Enable Forwarding and thus IPMASQ`" comment lines. It should also be noted that you **DO NOT** have to enable the "HTTPD" rules under the "Optional 'INPUT' section" for 2.4-based kernels. With 2.4 kernels, those lines are ONLY used if you want to allow HTTP traffic to terminate on the MASQ server's own local httpd process (usually Apache).

PORTFW FTP: If you have the "ip_conntrack_ftp" and "ip_nat_ftp" kernel modules loaded into kernel space (as already done in the rc.firewall-iptables script), the simple PREROUTING command like the one shown above changed for for port "21" should do the trick. This is much easier than the configuration for the older IPCHAINS / IPFWADM tools for the 2.2.x / 2.0.x kernels!

Please note, if you setup PORTFW to redirect traffic to an internal FTP server that is running on a NON-standard FTP port, say port 8021 instead of the usual port 21, you MUST tell the "ip_conntrack_ftp" module to be aware of the new FTP port. The reason for this configuration change is that FTP is not a NAT-friendly protocol. By telling the FTP NAT module about this non-standard FTP port, the NAT module and do it's job again. To enable this, edit your rc.firewall-* file and change the loading of the FTP module to look something like this:

```
/sbin/insmod ip_conntrack_ftp ports=21,8021
```

PORTFW Redirection of Internal requests:

In the past, if users PORTFWed port 80 on their EXTERNAL IP address to some internal machine, only the machines out on the Internet would be able to properly reach this internal WWW server. If you tried to contact this internal WWW server via the MASQ server's EXTERNAL address, it would fail. Fortunately, there are workarounds for all Linux kernels. IPTABLES for the 2.6.x and 2.4.x kernels supports this functionality natively. IPCHAINS support for the 2.2.x kernels and IPFWADM support for the 2.0.x kernels need to use the external REDIR tool (not currently covered in the HOWTO). For those of you who understand IPTABLES fairly well, the rule of thumb is a PREROUTING/DNAT rule is for enabling PORTFW from *external* machines and a POSTROUTING/SNAT rule is for enabling PORTFW from *internal* machines.

To support redirection like this from an internal host, add a rule like the one shown below ABOVE the "Catch all" FORWARDing rule in the rc.firewall-* file. The following example will REDIRECT all external as well as internal WWW traffic to the internal machine noted as PORTFWIP (192.168.0.10). This traffic will have the source IP address of the IP Masq server's internal IP address. Please change the IP address of the PORTFWIP variable to reflect your specific configuration:

```
#The following rule should be configured in *addition* to the above rule
# for enabling external to internal PORTFWing

$IPTABLES -t nat -A POSTROUTING -d $PORTFWIP -s $INTLAN -p tcp \
--dport 80 -m state --state NEW,ESTABLISHED,RELATED -j SNAT \
--to $INTIP
```

6.7.2. IPMASQADM-based PORTFWD'ing: Using IPMASQADM with 2.2.x kernels

First, make sure you have the newest 2.2.x kernel uncompressed into /usr/src/kernel/linux. If you haven't already done this, please see Section 3.2.2 section for full details. Next, download the "ipmasqadm.c" program from Section 2.7 into the /usr/src/kernel directory.

Next, you'll need to compile the 2.2.x kernel as shown in Section 3.2.2 section. Be sure to say "YES" to the IPPORTFW option when you configure the kernel. Once the kernel compile is complete and you have rebooted, return to this section.

Now, compile and install the IPMASQADM tool:

```
cd /usr/src
tar xzvf ipmasqadm-x.tgz
cd ipmasqadm-x
make
make install
```

Now, for this example, we are going to allow ALL WWW Internet traffic (port 80) hitting your Internet TCP/IP address to be forwarded to the internal Masqueraded machine at IP address 192.168.0.10.

PORTFW FTP: As mentioned above, there are two solutions for forwarding FTP server traffic to an internal MASQed PC. The first solution *IS* a BETA level **IP_MASQ_FTP** module for 2.2.x kernels to PORT Forward FTP connections to an internal MASQed FTP server. The other method is using a FTP proxy program (the URL is in Section 2.7. It should also be noted that the FTP kernel module also supports the adding of additional PORTFW FTP ports on the fly without the requirement of unloading and reloading the IP_MASQ_FTP module and thus breaking any existing FTP transfers. You can find more about this new code at the IPMASQ WWW site at http://ipmasq.webhop.net;. There are also examples and some additional information about PORTFWed FTP connection below in the 2.0.x. kernel section.

NOTE: Once you enable a port forwarder on port 80, that port can no longer be used by the Linux IP Masquerade server. To be more specific, if you have a WWW server already running on the MASQ server, a port forward will now give all Internet users the WWW pages from the -INTERNAL- WWW server and not the pages on your IP MASQ server.

Anyway, to enable port forwarding for HTTPd:

- Edit the /etc/rc.d/rc.firewall-* ruleset and ENABLE the "Optional" "HTTP" sections in both the INPUT and OUTPUT subsections.
- Add the following lines shown below JUST BELOW the "`ipchains -P forward DENY`" rule (in the "Optional FORWARD section"). Be sure to replace the "$EXTIP" variable's contents with your EXTERNAL Internet IP address on the IPMASQ server.

NOTE #2: If you get a dynamically assigned TCP/IP address from your ISP (PPP, DSL, Cablemodems, etc.), you **CANNOT load** this strong ruleset upon booting. You will either need to reload this firewall ruleset EVERY TIME you get a new IP address or make your /etc/rc.d/rc.firewall-ipchains-stronger ruleset more intelligent. To do this for various types of connections such as PPP or DHCP users, please see the Section 7.8 FAQ entry for all the details.

/etc/rc.d/rc.firewall-*

```
#echo "Enabling IPPORTFW Redirection on the external LAN.."
#
#   This will forward ALL port 80 traffic from the external IP address
#   to port 80 on the 192.168.0.10 machine
#
PORTFWIP="192.168.0.10"

/usr/sbin/ipmasqadm portfw -f
/usr/sbin/ipmasqadm portfw -a -P tcp -L $EXTIP 80 -R $PORTFWIP 80
```

That's it! Just re-run your /etc/rc.d/rc.firewall-* ruleset and test it out!

If you get the error message "ipchains: setsockopt failed: Protocol not available", you AREN'T running your new IPPORTFW enabled kernel. Make sure that you moved the new kernel over, re-run LILO, and then reboot again. If you are sure you are running your new kernel, run the command "ls /proc/net/ip_masq" and make sure the "portfw" file exists. If it doesn't, you must have made an error when configuring your kernel. Try again.

PORTFW Redirection of Internal requests:

It should be mention that this IPMASQ HOWTO currently does *NOT* provide any explination or examples on how to use the REDIR tool. If you need help setting it up for 2.2.x kernels, send me an email. For those who want to understand why PORTFW cannot redirect traffic for both external and internal interfaces (what the REDIR tool fixes), here is an email from Juanjo that better explains it.

```
From Juanjo Ciarlante
--

>If I use:
>
> ipmasqadm portfw -a -P tcp  -L 1.2.3.4 80 -R 192.168.2.3 80
>
>Everything works great from the outside but internal requests for the same
>1.2.3.4 address fail. Are there chains that will allow a machine on
localnet
>192.168.2.0 to accesss www.periapt.com without using a proxy?

Actually not.

I usually setup a ipmasqadm rule for outside, *AND* a port
redirector for inside. This works because ipmasqadm hooks before
redir will get the eventual outside connection, _but_ leaves things
ok if not (stated by APPROPIATE rules).

The actual "conceptual" problem comes from the TRUE client (peer) IP
goal (thanks to masq) being in the same net as target server.

The failing scenario for "local masq" is :
   client: 192.168.2.100
   masq:   192.168.2.1
   serv:   192.168.2.10

1)client->server packet
```

```
 a) client:  192.168.2.100:1025  -> 192.168.2.1:80   [SYN]
 b) (masq):  192.168.2.100:1025  -> 192.168.2.10:80  [SYN]
           (and keep  192.168.2.1:61000 192.168.2.100:1025 related)
 c) serv:    gets masqed packet (1b)

2)server->client packet
 a) serv:    192.168.2.10:80     -> 192.168.2.100:1025  [SYN,ACK]
 b) client:  192.168.2.100:1025  -> 192.168.2.10:80     [RST]

Now take a moment to compare (1a) with (2a).
You see, the server replied DIRECTLY to client bypassing masq (not
letting masq to UNDO the packet hacking) because it is in SAME net, so
the client resets the connection.

hope I helped.

Warm regards
Juanjo
```

6.7.3. IPPORTFW-based PORTFWD'ing: Using IPPORTFW on 2.0.x kernels

First, make sure you have the newest 2.0.x kernel uncompressed into /usr/src/kernel. If you haven't already done this, please see Section 3.2.3 for full details. Next, download the "ipportfw.c" program and the "subs-patch-x.gz" kernel patch from Section 3.2.3 into the /usr/src/ directory.

NOTE: Please replace the "x" in the "subs-patch-x.gz" file name with the most current version available on the site.

Next, if you plan on port forwarding FTP traffic to an internal server, you will have to apply an additional **NEW** *IP_MASQ_FTP* module patch found in Section 3.2.3. More details regarding this are later in this section. Please note that this is NOT the same patch as for the 2.2.x kernels so some functionality such as the dynamic FTP PORT functionality is not present.

Now, copy the IPPORTFW patch (subs-patch-x.gz) into the Linux directory

```
        cp /usr/src/subs-patch-1.37.gz /usr/src/linux
```

Next, apply the kernel patch to create the IPPORTFW kernel option:

```
        cd /usr/src/linux
        zcat subs-patch-1.3x.gz | patch -p1
```

Ok, time to compile the kernel as shown in Section 3.2.3. Be sure to say YES to the IPPORTFW option now available when you configure the kernel. Once the compile is complete and you have rebooted, return to this section.

Now with a newly compiled kernel, please compile and install the actual "IPPORTFW" program

```
cd /usr/src
gcc ipportfw.c -o ipportfw
mv ipportfw /usr/local/sbin
```

Now, for this example, we are going to allow ALL WWW Internet traffic (port 80) hitting your Internet TCP/IP address to then be forwarded to the internal Masqueraded machine at IP address 192.168.0.10.

NOTE: Once you enable a port forwarder on port 80, that port can no longer be used by the Linux IP Masquerade server. To be more specific, if you have a WWW server already running on the MASQ server and then you port forward port 80 to an internal MASQed computer, ALL internet users will see the WWW pages pages from the -INTERNAL- WWW server and not the pages on your IP MASQ server. This only performs a port forward to some other port, say 8080, to your internal MASQ machine. Though this will work, all Internet users will have to append **:8080** to the URL to then contact the internal MASQed WWW server.

Anyway, to enable port forwarding, edit the **/etc/rc.d/rc.firewall-*** ruleset. Add the follow lines but be sure to replace the word "$extip" with your Internet IP address.

NOTE #2: If you get a dynamically assigned TCP/IP address from your ISP (PPP, DSL, Cablemodems, etc.), you **CANNOT load** this strong ruleset upon booting. You will either need to reload this firewall ruleset EVERY TIME you get a new IP address or make your /etc/rc.d/rc.firewall-ipchains-stronger ruleset more intelligent. To do this for various types of connections such as PPP or DHCP users, please see the Section 7.8 FAQ entry for all the details.

/etc/rc.d/rc.firewall-*

```
#echo "Enabling IPPORTFW Redirection on the external LAN.."
#
#   This will forward ALL port 80 traffic from the external IP address
#   to port 80 on the 192.168.0.10 machine
#
/usr/local/sbin/ipportfw -C
/usr/local/sbin/ipportfw -A -t$extip/80 -R 192.168.0.10/80
```

That's it! Just re-run your /etc/rc.d/rc.firewall-* ruleset and test it out!

If you get the error message "ipfwadm: setsockopt failed: Protocol not available", you AREN'T running your new kernel. Make sure that you moved the new kernel over, re-run LILO, and then reboot again.

Port Forwarding FTP servers:

If you plan on port forwarding FTP to an internal machine, things get more complicated. The reason for this is because the standard **IP_MASQ_FTP** kernel module wasn't written for this though some users report that it works without any problems. Personally, without the patch, I've heard that extended file transfers in excess of 30 minutes will fail without the patch while other users swear that it works flawlessly. Anyway, I recommend that you try the following PORTFW instruction with the STOCK ip_masq_ftp module and see if it works for you. If it doesn't, try using the modified ip_masq_ftp module.

For those who need the module, Fred Viles wrote a modified IP_MASQ_FTP module to make things work. If you are curious what EXACTLY are the issues, download the following archive since Fred documents it quite well. Also understand that this patch is somewhat experimental and should be treated as such. It should be noted that this patch is ONLY available for the 2.0.x kernels though there is a different patch available for 2.2.x kernels.

So, to get the 2.0.x patch working, you need to:

- FIRST, apply the IPPORTFW kernel patch as shown earlier in this section.
- Download the "msqsrv-patch-36" from Fred Viles's FTP server in Section 3.2.3and put it into /usr/src/linux.
- Patch the kernel with this new code by running "cat msqsrv-patch-36 | patch -p1"
- Next, replace the original **"ip_masq_ftp.c"** kernel module with the new one

  mv /usr/src/linux/net/ipv4/ip_masq_ftp.c /usr/src/linux/net/ipv4/ip_masq_ftp.c.orig

  mv /usr/src/linux/ip_masq_ftp.c /usr/src/linux/net/ipv4/ip_masq_ftp.c
- Lastly, build and install the kernel with this new code in place.

Once this is complete, edit the /etc/rc.d/rc.firewall-* ruleset and add the following lines, but be sure to replace the word "$extip" with your Internet IP address.

This example, like the one above, will allow ALL FTP Internet traffic (port 21) hitting your Internet TCP/IP address to then be forwarded to the internal Masqueraded machine at IP address 192.168.0.10.

NOTE: Once you enable a port forwarder on port 21, that port can no longer be used by the Linux IP Masquerade server. To be more specific, if you have an FTP server already running on the MASQ server, a port forward will now give all Internet users the FTP files from the -INTERNAL- FTP server and not the files on your IP MASQ server.

/etc/rc.d/rc.firewall-*

```
#echo "Enabling IPPORTFW Redirection on the external LAN.."
#
/usr/local/sbin/ipportfw -C
/usr/local/sbin/ipportfw -A -t$extip/21 -R 192.168.0.10/21

#NOTE: If you are using multiple local port numbers to PORTFW
#      to multuple internal FTP servers (say, 21, 2121, 2112,
#      etc, you need to configure the ip_masq_ftp nodule to
#      listen to these ports.  To do this, edit the
#      /etc/rc.d/rc.firewall-* script as shown in this HOWTO
#      to look like:
#
# /sbin/modprobe ip_masq_ftp ports=21,2121,2112
#
# Re-run the /etc/rc.d/rc.firewall-* script for these changes to
# take effect.


#Please note that PORTFWing port 20 is probably NOT required
#  for ACTIVE connections as the internal FTP server will
#  initiate this port 20 connection and it will be properly
#  handled by the classic MASQ mechanisms.
```

That's it! Just re-run your /etc/rc.d/rc.firewall-* ruleset and test it out!

PORTFW Redirection of Internal requests:

It's not clear if the REDIR tool will compile or work against legacy 2.0.x kernels. It should also be mentioned that this IPMASQ HOWTO currently does *NOT* provide any explination or examples on how to use the REDIR tool. If you need help setting it up,

send me an email. If you would like to learn more about REDIR, please see the above section for the 2.2.x kernels.

6.8. CU-SeeMe and Linux IP-Masquerade

Linux IP Masquerade supports CuSeeme via the **"ip_masq_cuseeme"** kernel module. This kernel modules should be loaded in the /etc/rc.d/rc.firewall-* script. Once the "ip_masq_cuseeme" module is installed, you should be able to both initiate and receive CuSeeme connections to remote reflectors and/or users.

NOTE: It is recommended to use the IPPORTFW tool instead of the old IPAUTOFW tool for running CuSeeme.

If you need more explicit information on configuring CuSeeme, see Michael Owings's CuSeeMe page for a Mini-HOWTO or The IP Masquerade Resources for a mirror of the Mini-HOWTO.

6.9. Mirabilis ICQ

ICQ, the instant messaging client now owned by AOL, has changed over the years. All modern ICQ clients are NAT friendly and thus DON'T require any special NAT modules, PORTFW tricks, etc.

IF, for some reason, you want to run an OLD ICQ client, you can read this section. If not, just IGNORE all this info. I am leaving this in the HOWTO demonstrate large a LARGE PORTFW example.

There are three methods of getting ICQ to work behind a Linux MASQ server. These solutions include the use the ICQ Masq module (for 2.2.x and 2.0.x kernels), using IPPORTFW for basic ICQ functionality, or setting up a SOCKS proxy server.

MODULE: The ICQ module was written for the older generation of ICQ clients for both the 2.2.x and 2.0.x kernels. This module allows for the simple setup of multiple ICQ users behind a MASQ server. It also doesn't require any special changes to the ICQ client (s). Recently, AOL changed the protocol and ports used for ICQ. Because of this, many users might find that the ip_masq_icq module will no longer help them. For users of the older ICQ clients, the 2.2.x version of the module supports file transfer and read-time chat. The 2.0.x kernel module doesn't support file transfers and there is no module available for the 2.4.x kernels.

PORTFW: Your next option is to use port forwarding. With port forwarding, basic ICQ chat will work but file transfers might not be very reliable. Please see below for an example of how to configure ICQ PORTFW.

SOCKS: Finally, your last and possibly best option is to setup a SOCKS proxy server on your Linux machine. This service can happily co-exist with the MASQ service and ICQ should be fully functional regardless of what Linux kernel version you are running. The use of a SOCKS server will require ALL ICQ clients to be reconfigured to use it and the installation and configuration of a SOCKS server has nothing to do with IP Masquerade. Because of this, SOCKS is not covered in this HOWTO.

If you are interested in Andrew Deryabin's djsf@usa.net ICQ IP Masq module for the 2.2.x and 2.0.x kernels, please see Section 2.7 for details.

To use port forwarding (PORFW)for ICQ, you will have to make some changes on both Linux and ICQ clients but all ICQ messaging, URLs, chat, and some file transfers should work.

- First, you need to be running a Linux kernel with IPPPORTFW enabled. Please see Section 6.7for more details.
- Next, you need to add the following lines to your /etc/rc.d/rc.firewall-* file. This example assumes that 10.1.2.3 is your external Internet IP address and your internal MASQed ICQ machine is 192.168.0.10:

- The following example is for a 2.2.x kernel with IPCHAINS:

  I have included two examples here for the user: Either one would work fine:

  Example #1

```
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2000 -R 192.168.0.10 2000
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2001 -R 192.168.0.10 2001
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2002 -R 192.168.0.10 2002
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2003 -R 192.168.0.10 2003
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2004 -R 192.168.0.10 2004
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2005 -R 192.168.0.10 2005
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2006 -R 192.168.0.10 2006
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2007 -R 192.168.0.10 2007
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2008 -R 192.168.0.10 2008
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2009 -R 192.168.0.10 2009
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2010 -R 192.168.0.10 2010
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2011 -R 192.168.0.10 2011
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2012 -R 192.168.0.10 2012
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2013 -R 192.168.0.10 2013
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2014 -R 192.168.0.10 2014
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2015 -R 192.168.0.10 2015
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2016 -R 192.168.0.10 2016
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2017 -R 192.168.0.10 2017
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2018 -R 192.168.0.10 2018
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2019 -R 192.168.0.10 2019
/usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 2020 -R 192.168.0.10 2020
```

  Example #2

```
port=2000
while [ $port -le 2020 ]
  do
    /usr/local/sbin/ipmasqadm portfw -a -P tcp -L 10.1.2.3 $port -R 192.168.0.10
$port
    port=$((port+1))
done
```

- The following example is for a 2.0.x kernel with IPFWADM:

  I have included two examples here for the user: Either one would work fine:

  Example #1

```
/usr/local/sbin/ipportfw -A -t10.1.2.3/2000 -R 192.168.0.10/2000
/usr/local/sbin/ipportfw -A -t10.1.2.3/2001 -R 192.168.0.10/2001
/usr/local/sbin/ipportfw -A -t10.1.2.3/2002 -R 192.168.0.10/2002
/usr/local/sbin/ipportfw -A -t10.1.2.3/2003 -R 192.168.0.10/2003
/usr/local/sbin/ipportfw -A -t10.1.2.3/2004 -R 192.168.0.10/2004
/usr/local/sbin/ipportfw -A -t10.1.2.3/2005 -R 192.168.0.10/2005
/usr/local/sbin/ipportfw -A -t10.1.2.3/2006 -R 192.168.0.10/2006
/usr/local/sbin/ipportfw -A -t10.1.2.3/2007 -R 192.168.0.10/2007
/usr/local/sbin/ipportfw -A -t10.1.2.3/2008 -R 192.168.0.10/2008
/usr/local/sbin/ipportfw -A -t10.1.2.3/2009 -R 192.168.0.10/2009
/usr/local/sbin/ipportfw -A -t10.1.2.3/2010 -R 192.168.0.10/2010
/usr/local/sbin/ipportfw -A -t10.1.2.3/2011 -R 192.168.0.10/2011
/usr/local/sbin/ipportfw -A -t10.1.2.3/2012 -R 192.168.0.10/2012
/usr/local/sbin/ipportfw -A -t10.1.2.3/2013 -R 192.168.0.10/2013
/usr/local/sbin/ipportfw -A -t10.1.2.3/2014 -R 192.168.0.10/2014
/usr/local/sbin/ipportfw -A -t10.1.2.3/2015 -R 192.168.0.10/2015
/usr/local/sbin/ipportfw -A -t10.1.2.3/2016 -R 192.168.0.10/2016
/usr/local/sbin/ipportfw -A -t10.1.2.3/2017 -R 192.168.0.10/2017
/usr/local/sbin/ipportfw -A -t10.1.2.3/2018 -R 192.168.0.10/2018
/usr/local/sbin/ipportfw -A -t10.1.2.3/2019 -R 192.168.0.10/2019
/usr/local/sbin/ipportfw -A -t10.1.2.3/2020 -R 192.168.0.10/2020
```

Example #2

```
port=2000
while [ $port -le 2020 ]
  do
    /usr/local/sbin/ipportfw -A t10.1.2.3/$port -R 192.168.0.10/$port
    port=$((port+1))
done
```

- Once your new rc.firewall-* is ready, reload the ruleset to make sure things are OK by simply typing in "/etc/rc.d/rc.firewall-*". If you get any errors, you either don't have IPPORTFW support in the kernel or you made a typo in the rc.firewall file.
- Now, in ICQ's Preferences-->Connection, configure it to be "Behind a LAN" and "Behind a firewall or Proxy". Now, click on "Firewall Settings" and configure it to be "I don't use a SOCK5 proxy". Also note that it was previously recommended to change ICQ's "Firewall session timeouts" to "30" seconds BUT many users have found that ICQ becomes unreliable. It has been found that ICQ is more reliable with its stock timeout setting (don't enable that ICQ option) and simply change MASQ's timeout to 160 seconds. You can see how to change this timeout in Section 3.4.3 and Section 3.4.2 rulesets. Finally, click on Next and configure ICQ to "Use the following TCP listen ports.." from "2000" to "2020". Now click done.

  Now ICQ will tell you that you will have to restart ICQ for the changes to take effect. To be honest, I had to REBOOT the Windows9x machine in order for things to work right but some users might say otherwise. So.. try it both ways.
- A user once told me that by simply portforwarding port 4000 to his ICQ machine, it worked perfectly. He reported that EVERYTHING worked fine (even chat, file transfers, etc) WITHOUT re-configuring ICQ from its default settings. Your mileage might vary on this topic but I thought you might like to hear about this alternative configuration.

6.10. Gamers: The LooseUDP patch

The LooseUDP patch allows semi-NAT-friendly games that usually use UDP connections to both WORK behind a Linux IP Masquerade server.

What the LooseUDP patch does is allow ALL UDP packets to be NATed through the MASQ box without any checks or expiration. This liberal forwarding method is considered insecure by many and is disabled in modern 2.2.x kernels. The 2.4.x kernels with it's IPTABLES stateful UDP inspection only allows incoming UDP packets into the machine (and thus MASQ) if there was already an outbound UDP packet to that same host in it's stateful table. If the MASQ host hasn't sent a UDP packet to the remote host within ~30 seconds, the return UDP table entry is deleted. Because of this, IPTABLES removes most of the need for the LooseUDP patch as it does it in a more secure fashion.

Currently, LooseUDP is available as a patch for 2.0.36+ kernels and it is already built into 2.2.3+ kernels though it is now DISABLED by DEFAULT in 2.2.16+ (please see the example rc.firewal ruleset comments for details).

To get LooseUDP running on a 2.0.x kernel, follow the following steps:

- Have the newest 2.0.x kernel sources uncompressed in the /usr/src/linux directory
- ABSOLUTELY REQUIRED for v2.0.x: Download and install the IPPORTFW patch from Section 3.2.3and as described in Section 6.7of the HOWTO.
- Download the LooseUDP patch from Section 3.2.3

  Now, put the LooseUDP patch in the /usr/src/linux directory. Once this is done, type in:

```
For a compressed patch file:  zcat loose-udp-2.0.36.patch.gz | patch -p1
```

```
For a NON-compressed patch file:  cat loose-udp-2.0.36.patch | patch -p1
```

  Now, depending on the version of your "patch", You will then see the following text:

```
patching file `CREDITS'
patching file `Documentation/Configure.help'
patching file `include/net/ip_masq.h'
patching file `net/ipv4/Config.in'
patching file `net/ipv4/ip_masq.c'
```

- If you see the text "Hunk FAILED" only ONCE and ONLY ONCE at the very beginning of the patching, don't be alarmed. You probably have an old patch file (this has been fixed) but it still works. If it fails completely, make sure you have applied the IPPORTFW kernel patch FIRST.

  Once the patch is installed, re-configure the kernel as shown in Section 3.2.3 and be sure to say "Y" to the "IP: loose UDP port managing (EXPERIMENTAL) (CONFIG_IP_MASQ_LOOSE_UDP) [Y/n/?]" option.

To get LooseUDP running on a 2.2.x kernel, follow the following steps:

- In the /etc/rc.d/rc.firewall-* script, goto the BOTTOM of the file and find the LooseUDP section. Change the "0" in the line: `echo "0" > /proc/sys/net/ipv4/ip_masq_udp_dloose` to a "1" and re-run the rc.firewall-* ruleset. An example of this is given in both Section 3.4.2 example and Section 6.4.3 example.

NOTE: The LooseUDP code is /not/ available (?required?) for the 2.4.x kernels

- See the begining of this section for all the details. Basically, the old 2.0.x / 2.2.x LooseUDP code was considered a security issue. Because of this, it was removed from the kernel. Fortunately, some games that used to require the LooseUDP code on the 2.2.x IPCHAINS system might work just final under the 2.4.x IPTABLES kernels. If the games don't work, I'm not aware of a solution for you. Sorry.

Once you are running the new LooseUDP enabled kernel, you should be good to go for most NAT-friendly games. Some URLs have been given for patches to make games like BattleZone and others NAT friendly. Please see Section 6.3.1 for more details.

Chapter 7. Frequently Asked Questions

If you can think of any useful FAQ suggestions, please send it to dranch@trinnet.net. Please clearly state the question and an appropriate answer (if you have it). Thank you!

7.1. (Distro) - What Linux Distributions support IP Masquerading?

If your Linux distribution doesn't support IP MASQ out of the box, don't worry. All you have to do is to re-compile the kernel as shown above in this HOWTO.

NOTE: If you can help us fill out this table, please email dranch@trinnet.net.

- Caldera < v1.2 : NO - ?
- Caldera v1.3 : YES - 2.0.35 based
- Caldera v2.2 : YES - 2.2.5 based
- Caldera eServer v2.3 : YES - ? based
- Debian v1.3 : NO - ?
- Debian v2.0 : NO - ?
- Debian v2.1 : YES - 2.2.1 based
- Debian v2.2 : YES - 2.2.15 based
- Debian v3.0 : YES - 2.4.18 based
- DLX Linux v? : ? - ?
- DOS Linux v? : ? - ?
- FloppyFW v1.0.2 : ? - ?
- Gentoo v1.4 : YES - ? Based
- Hal91 Linux v? : ? - ?
- Linux PPC vR4 : NO - ?
- Linux Pro v? : ? - ?
- LinuxWare v? : ? - ?
- Mandrake v5.3 : YES - ?
- Mandrake v6.0 : YES - 2.2.5
- Mandrake v6.1 : YES - ?
- Mandrake v7.0 : YES - 2.2.14
- Mandrake v7.1 : YES - 2.2.15
- Mandrake v7.2 : YES - 2.2.17
- Mandrake v8.0 : YES - ?

- Mandrake v8.1 : YES - 2.4.8
- Mandrake v8.2 : YES - ?
- Mandrake v9.0 : YES - ?
- Mandrake v9.1 : YES - 2.4.21-pre
- MkLinux v? : ? - ?
- MuLinux v3rl : YES - ?
- Redhat < v4.x : NO - ?
- Redhat v5.0 : YES - ?
- Redhat v5.1 : YES - 2.0.34 based
- Redhat v5.2 : YES - 2.0.36 based
- Redhat v6.0 : YES - 2.2.5 based
- Redhat v6.1 : YES - 2.2.12 based
- Redhat v6.2 : YES - 2.2.14 based
- Redhat v7.0 : YES - 2.2.16 based
- Redhat v7.2 : YES - 2.4.7 based
- Redhat v7.3 : YES - 2.4.? based
- Redhat v8.0 : YES - 2.4.18? based
- Redhat v9.0 : YES - 2.4.20 based
- Slackware v3.0 : ? - ?
- Slackware v3.1 : ? - ?
- Slackware v3.2 : ? - ?
- Slackware v3.3 : ? - 2.0.34 based
- Slackware v3.4 : ? - ?
- Slackware v3.5 : ? - ?
- Slackware v3.6 : ? - ?
- Slackware v3.9 : ? - 2.0.37pre10 based
- Slackware v4.0 : YES - 2.2.6 based
- Slackware v7.0 : YES - 2.2.13 based
- Slackware v7.1 : YES - 2.2.16 based
- Slackware v8.0 : YES - 2.4.17 based
- Slackware v8.1 : YES - ? based
- Stampede Linux v? : ? - ?
- SuSE v5.2 : YES - 2.0.32 base
- SuSE v5.3 : YES - ?
- SuSE v6.0 : YES - 2.0.36 based
- SuSE v6.1 : YES - 2.2.5 based
- SuSE v6.3 : YES - 2.2.13 based
- Tomsrbt Linux v? : ? - ?
- TurboLinux Lite v4.0 : YES - ?
- TurboLinux v6.0 : YES - 2.2.12 based
- TriLinux v? : ? - ?
- Yggdrasil Linux v? : ? - ?

7.2. (Requirements) - What are the minimum hardware requirements and any limitations for IP Masquerade? How well does it perform?

A 486/66 box with 16MB of RAM was more than sufficient to fill a 1.54Mb/s T1 100%! MASQ has also been known to run quite well on 386SX-16s with 8MB of RAM. Yet, it should be noted that Linux IP Masquerade starts thrashing the system with more than 500 MASQ entries.

The only application that I know which can temporarily break Linux IP Masquerade, is GameSpy. Why? When it refreshes its lists, it creates 10,000s of quick connections in a VERY short period of time. Until these sessions timeout, the MASQ tables become "FULL". See Section 7.23 of the FAQ for more details.

While we are at it:

There is a hard limit of 4096 concurrent connections each for TCP & UDP. This limit can be changed by fiddling the values in **/usr/src/linux/net/ipv4/ip_masq.h** - a maximum limit of 32000 should by OK. If you want to change the limit - you need to change the PORT_MASQ_BEGIN & PORT_MASQ_END values to get an appropriately sized range above 32K and below 64K.

7.3. (Errors) - When I run my specific rc.firewall-* ruleset, I get "command not found" errors. Why?

First off, when I say rc.firewall-*, what I really mean is to use one of the various types of firewall rulesets depending on what kernel version you're running. Your options from this HOWTO include: rc.firewall-iptables or rc.firewall-iptables-stronger or rc.firewall-ipchains or rc.firewall-ipchains-stronger or rc.firewall-ipfwadm or rc.firewall-ipfwadm-stronger.

Next, how did you put the rc.firewall-* onto your machine? Did you cut&paste it into a TELNET window, FTP it from a Windows/DOS machine, etc? Try this.. log into your Linux box and run "vim -b /etc/rc.d/rc.firewall-*" and see if all your lines end in a ^M. If they do, delete all the ^M and try again.

7.4. (Still wont work) - I've checked all my configurations, I still can't get IP Masquerade to work. What should I do?

- Stay calm. Get yourself a cup of tea, coffee, soda, etc., and have a rest. Once your mind is clear, try the suggestions mentioned below. Setting up Linux IP Masquerading is NOT hard but there are several concepts that will be new to you.
- Again, go through all the steps in Chapter 5. 99% of all first-time Masquerade users who have problems haven't looked here.
- Check the IP Masquerade Mailing List Archives, most likely your questions or problems are a common one and can be found in a simple Archive search.
- Check out the TrinityOS document. It covers IP Masquerading for both the 2.0.x and 2.2.x kernels and MANY other topics including PPPd, DialD, DHCP, DNS, Sendmail, etc.
- Make sure that you aren't running ROUTED or GATED. To verify, run "ps aux | grep -e routed -e gated"
- Post your question to the IP Masquerade Mailing List (see next the FAQ section for details). Please only use this if you cannot find the answer from the IP Masquerading Archive. Be sure to include all the information requested in Chapter 5 in your email!!
- Post your question to a related Linux NNTP newsgroup.
- Send email to ambrose@writeme.com and dranch@trinnet.net. You have a better chance of getting a reply from the IP Masquerading Email list than either of us.
- Check your configurations again :-)

7.5. (Email list) - How do I join or view the IP Masquerade and/or IP Masqurade Developers mailing lists and archives?

There are two ways to join the two Linux IP Masquerading mailing lists. The first way is to send an email to masq-request@indyramp.com. To join the Linux IP Masquerading Developers mailing list, send an email to masq-dev-request@indyramp.com. Please see the bullet below for more details.

- Subscribe via email: Now put the word "subscribe" in either the subject or body of the e-mail message. If you want to only subscribe to the Digest version of either the main MASQ or MASQ-DEV list (all e-mails on the given list during the week are sent to you in one big email), put the words "subscribe digest" in either the subject or body of the e-mail message.

  Once the server receives your request, it will subscribe you to your requested list and give you a PASSWORD. Save this password as you will need it to later unsubscribe from the list or change your options.

The second method is to use a WWW browser and subscribe via a form at http://home.indyramp.net/mailman/listinfo/masq for the main MASQ list or http://home.indyramp.net/mailman/listinfo/masq-dev for the MASQ-DEV list.

Once subscribed, you will get emails from your subscribed list. It should be also noted that both subscribed and NON-subscribed users can access the two list's archives. To do this, please see the above two WWW URLs for more details.

Lastly, please note that you can only post to the MASQ list from the original account/address you used to subscribe.

If you have any problems regarding the mailing list or the mailing list archive, please contact Robert Novak.

7.6. (NAT vs. Proxy) - How does IP Masquerade differ from Proxy or NAT services?

```
Proxy:  Proxy servers are available for: Win95, NT, Linux, Solaris, etc.

            Pro:    + (1) IP address ; cheap
                    + Optional caching for better performance (WWW, etc.)

            Con:    - All applications behind the proxy server must both SUPPORT
                      proxy services (SOCKS) and be CONFIGURED to use the Proxy
                      server
                    - Screws up WWW counters and WWW statistics

          A proxy server uses only (1) public IP address, like IP MASQ, and acts
          as a translator to clients on the private LAN (WWW browser, etc.).
          This proxy server receives requests like TELNET, FTP, WWW,
          etc. from the private network on one interface.  It would then in turn,
          initiate these requests as if someone on the local box was making the
          requests.   Once the remote Internet server sends back the requested
          information, it would re-translate the TCP/IP addresses back to the
          internal MASQ client and send traffic to the internal requesting host.
          This is why it is called a PROXY server.

                 Note:  ANY applications that you might want to use on the
                         internal machines *MUST* have proxy server support
                         like Netscape and some of the better TELNET and FTP
                         clients.  Any clients that don't support proxy servers
```

won't work.

Another nice thing about proxy servers is that some of them can also do caching (Squid for WWW). So, imagine that you have 50 proxied hosts all loading Netscape at once. If they were installed with the default homepage URL, you would have 50 copies of the same Netscape WWW page coming over the WAN link for each respective computer. With a caching proxy server, only one copy would be downloaded by the proxy server and then the proxied machines would get the WWW page from the cache. Not only does this save bandwidth on the Internet connection, it will be MUCH MUCH faster for the internal proxied machines.

MASQ: or 1:Many NAT

IP Masq is available on Linux and a few ISDN routers such as the Zytel Prestige128, Cisco 770, NetGear ISDN routers, etc.

Pro:
+ Only (1) IP address needed (cheap)
+ Doesn't require special application support
+ Uses firewall software so your network can become more secure

Con:
- Requires a Linux box or special ISDN router (though other products might have this..)
- Incoming traffic cannot access your internal LAN unless the internal LAN initiates the traffic or specific port forwarding software is installed. Many NAT servers CANNOT provide this functionality.
- Special protocols need to be uniquely handled by firewall redirectors, etc. Linux has full support for this (FTP, IRC, etc.) capabilty but many routers do NOT (NetGear DOES).

Masq or 1:Many NAT is similar to a proxy server in the sense that the server will perform IP address translation and fake out the remote server (WWW for example) as if the MASQ server made the request instead of an internal machine.

The major difference between a MASQ and PROXY server is that MASQ servers don't need any configuration changes to all the client machines. Just configure them to use the linux box as their default gateway and everything will work fine. You WILL need to install special Linux modules for things like RealAudio, FTP, etc. to work)!

Also, many users operate IP MASQ for TELNET, FTP, etc. *AND* also setup a caching proxy on the same Linux box for WWW traffic for the additional performance.

NAT:

NAT servers are available on Windows 95/NT, Linux, Solaris, and some of the better ISDN routers (not Ascend)

Pro:
+ Very configurable

```
                    + No special application software needed

            Con:    - Requires a subnet from your ISP (expensive)

        Network Address Translation is the name for a box that would have a pool of
        valid IP addresses on the Internet interface which it can use.  Whenever the
        Internal network wanted to go to the Internet, it associates an available
        VALID IP address from the Internet interface to the original requesting
        PRIVATE IP address.  After that, all traffic is re-written from the NAT
        public IP address to the NAT private address.  Once the associated PUBLIC
        NAT address becomes idle for some pre-determined amount of time, the
        PUBLIC IP address is returned back into the public NAT pool.

        The major problem with NAT is, once all of the free public IP addresses are
        used, any additional private users requesting Internet service are out of
        luck until a public NAT address becomes free.
```

For an excellent and very comprehensive description of the various forms of NAT, please see:

- http://www.suse.de/~mha/linux-ip-nat/diplom/nat.html/

Here is another good site to learn about NAT, although many of the URLs are old but still valid:

- http://www.linas.org/linux/load.html/

7.7. (GUI) - Are there any GUI firewall creation/management tools?

Yes! They vary in the type of user interface, complexity, etc. but they are quite good though most are only for the IPFWADM tool so far. Here is a short list of available tools in alphabetical order. If you know of any others or have any thoughts on which ones are good/bad/ugly, please email David.

- John Hardin's IPFWADM Dot file generator - a IPCHAINS version is in the works
- Sonny Parlin's fBuilder: From the author of FWCONFIG, this new solution is fully WWW based and offers redundancy options, etc for both IPCHAINS and NetFilter.
- William Stearns's Mason - A Build-a-ruleset on-the-fly type system

---

7.8. (MASQ and Dynamic IPs) - Does IP Masquerade work with dynamically assigned IP addresses?

Yes, it works with either dynamic IP addressed assigned by your ISP via either PPP or a DHCP server (common for Cablemodem and DSL users). As long as you have a valid Internet IP address, it should work. Of course, static IPs work too. If you plan on implementing a strong IPTABLES, IPCHAINS, or IPFWADM ruleset and/or plan on using a Port forwarder, your ruleset will have to be re-executed everytime your IP address changes.

So, re-running the rc-firewall-* ruleset really depends on which method you get your IP addresses:

- PPP: The /etc/ppp/ip-up script is always run when a PPP connection comes up. Because of this, we can make the rc.firewall-* go get the new PPP IP address and update the firewall ruleset. If the /etc/ppp/ip-up file doesn't exist or if does exists, simply edit that file and append a line containing the name of your chosen firewall ruleset. For example, to run the SIMPLE IPTABLES ruleset:

```
/etc/rc.d/rc.firewall-iptables
```

- DHCP: If you get your IP address via DHCP, common for Cablemodem and DSL users, it's easy get the rc.firewall-* ruleset to run when you get a new IP lease. How this happens depends on what DHCP client your distribution uses:

  - **dhclient** : Most modern Linux distributions use dhclient from ISC. To re-run your specific rc-firewall-* script when your system gets a new IP address, add the following line to the /etc/dhclient-exit-hooks file. Please note that this example is loging the SIMPLE IPTABLES ruleset. Please use the correct rc.firewall-* name for your specific setup:

    ```
    /etc/rc.d/rc.firewall-iptables
    ```

  - **pump** : Many older Redhat distributions use this DHCP client. To re-run the rc-firewall-* script when your system gets a new IP address, add the following line to the /etc/pump.conf file. Please note that this example is loading the SIMPLE IPTABLES ruleset. Please use the correct rc.firewall-* name for your specific setup:

    ```
    script /etc/rc.d/rc.firewall-iptables
    ```

  - **dhcpcd** : Many older distros use this DHCP client. To re-run the rc-firewall-* script when your system gets a new IP address depends on which verion of dhcpcd you're using.

    For newer dhcpcd client versions, append the following line to the /etc/dhcpcd-[interface].exe file. Please note that you have to replace the [interface] text with the name of your Interface connecting to the Internet. For this example, we are going run the SIMPLE IPTABLES ruleset against the eth0 interface by editing the /etc/dhcpc/dhcpcd-eth0.exe file:

    ```
    /etc/rc.d/rc.firewall-iptables
    ```

    For old dhcpcd client versions, you need to figure out what script starts up dhcpcd (depends on the Linux distribution and it's version). From there, you need to replace the specific dhcpcd line with the folling line with the correct Internet-facing interface name. For this example, dhcpcd will run the SIMPLE IPTABLES ruleset against the eth0 interface:

    ```
    dhcpcd -c /etc/rc.d/rc.firewall eth0
    ```

Please also see the top of TrinityOS - Section 10 for additional help with strong firewall rulesets and Dynamic IP addresses.

7.9. (MASQ and various networks) - Can I use a cable modem (both bi-directional and with modem returns), DSL, satellite link, etc. to connect to the Internet and use IP Masquerade?

Yes, as long as Linux supports that network interface, it should work. If you receive a dynamic IP address, please see the URL under the "Does IP Masquerade work with dynamically assigned IP" FAQ item above.

7.10. (Dial on Demand) - Can I use Diald or the Dial-on-Demand feature of PPPd with IP MASQ?

Definitely! IP Masquerading is totally transparent to Diald or PPP. The only thing that might become an issue is if you use STRONG firewall rulesets with dynamic IP addresses. See the FAQ item, "Does IP Masquerade work with dynamically assigned IP addresses?" above for more details.

7.11. (Apps) - What applications are supported with IP Masquerade?

It is very difficult to keep track of a list of "working applications". However, most of the normal Internet applications are supported, such as WWW browsing (Netscape, MSIE, etc.), FTP (such as WS_FTP), TELNET, SSH, RealAudio, POP3 (incoming email - Pine, Eudora, Outlook), SMTP (outgoing email), etc. A somewhat more complete list of MASQ-compatible clients can be found in Section 6.3 in this HOWTO.

Applications involving more complicated protocols or special connection methods such as video conferencing software need special helper tools.

For more details, please see the Linux IP masquerading Applications page.

7.12. (Distro Setup) - How can I get IP Masquerade running on Redhat, Debian, Slackware, etc.?

No matter which Linux distribution you have, the procedures for setting up IP Masquerade mentioned in this HOWTO should apply. Some distributions may have GUI or special configuration files that make the setup easier. We tried our best to write this HOWTO as general as possible.

7.13. (Timeouts) - Connections seem to break if I don't use

them often. Why is that?

IP Masq, by default, sets its timers for TCP session, TCP FIN, and UDP traffic to 15 minutes. It is recommend to use the following settings (as already shown in this HOWTO's /etc/rc.d/rc.firewall-* ruleset) for most users:

Linux 2.4.x with IPTABLES

```
IPMASQ timeouts are NOT adjustable under IPTABLES
```

Linux 2.2.x with IPCHAINS:

```
# MASQ timeouts
#
#   2 hrs timeout for TCP session timeouts
#  10 sec timeout for traffic after the TCP/IP "FIN" packet is received
#  60 sec timeout for UDP traffic (MASQ'ed ICQ users must enable a
30sec
#     firewall timeout in ICQ itself)
#
/ipchains -M -S 7200 10 60
```

Linux 2.0.x with IPFWADM:

```
# MASQ timeouts
#
#   2 hrs timeout for TCP session timeouts
#  10 sec timeout for traffic after the TCP/IP "FIN" packet is received
#  60 sec timeout for UDP traffic (MASQ'ed ICQ users must enable a
30sec
#     firewall timeout in ICQ itself)
#
/sbin/ipfwadm -M -s 7200 10 60
```

7.14. (Odd Behavior) - When my Internet connection first comes up, nothing works. If I try again, everything then works fine. Why is this?

The reason is because you have a dynamic IP address and when your Internet connection first comes up, IP Masquerade doesn't know its IP address. There is a solution to this. In your /etc/rc.d/rc.firewall-* ruleset, add the following:

```
# Dynamic IP users:
#
#   If you get your IP address dynamically from SLIP, PPP, or DHCP, enable this
#   following option.  This enables dynamic-ip address hacking in IP MASQ,
making
#   the life with Diald and similar programs much easier.
#
echo "1" > /proc/sys/net/ipv4/ip_dynaddr
```

7.15. (MTU) - IP MASQ seems to be working fine but some sites don't work. This usually happens with WWW and some FTP sites.

Depending on what Linux kernel version you are running on the MASQ server, some will people disagree on the real problem. The two following arguments have valid points, are inter-related, and users from each camp continue to debate this to this day.

- With modern 2.4.x Linux systems, most users point their finger at the adminstrators of these remote broken sites (typically SSL-encrypted WWW sites, etc.) or your MASQ server's upstream router run by your ISP. The main though it that these machines are either filtering or not properly responding to SOME or ALL FORMS of ICMP packets (specifically ICMP Code 3 Type 4 - Fragmentation Needed) messages due to a fray of misplaced security paranoia.

  What does that all mean? Basically, say your machine is connected to the Internet with a MTU of 1492 bytes (Maximum Transmission Unit -- the maximum packet size your computer can transmit) which is common for PPPoE users. At the same time, the remote WWW/FTP site is connected to the Internet at a MTU of 1500 bytes. The way that TCP/IP works is that when a TCP connection is being negotiated for your HTTP / FTP connection, the remote side will try to verify that a 1500 byte packet can reach your computer via the initial TCP "SYN" packet.

  Since the packet is too big for your connection, your upstream router (run by your ISP) will send a ICMP 3:4 (fragmentation needed) packet back to the remote WWW / FTP server. Within this packet is a recommended smaller MTU size to retry with. The problem is that either your local upstream router, some router between you and the remote server, or the remote HTTP / FTP server is either misconfigured or has a firewall in front of it that is BLOCKing these ICMP packets.
- The final UNCOMMON possibility is a debatable 2.0 / 2.2 kernel bug in the IPMASQ code. Some users point their finger to the fact that IPMASQ might have problems with ICMP packets that have the DF or "Don't Fragment" bit set. Basically, when a MASQ box connects to the Internet with an MTU of anything less than 1500, some packets will have the DF field set. Though changing the MTU of the MASQ server's Internet link to 1500 seemingly fixes the problem, the possible bug is still there. What is believed to be happening in these older kernels is that the MASQ code is not properly re-writing the return IP addresses of the ICMP 3 Sub 4 code packets back to the originating MASQed computer. Because of this, these critical packets get dropped.

No worries though. A there are several perfectly good ways to fix this nasty MTU problem:

- Enable PMTU clamping in PPPoE

  This solution is mostly for modern 2.4.x and 2.2.x kernel users connected to the Internet via a PPPoE DSL or Cablemodem connections. This solution allows for changes to be done ONLY on the MASQ server itself and not on all of the internal MASQ clients.
- Enable PMTU clamping via IPTABLES

This solution is only modern 2.4.x kernel users connected via ANY type of Internet connection. This solution allows for changes to be done ONLY on the MASQ server itself and not on all of the internal MASQ clients.

- Change your MASQ server's Internet Link MTU

  This solution will work for any Linux kernel version but is is NOT a solution if you have a PPPoE connection for DSL or Cablemodem users.

  It should be noted that some users will balk at this solution because it can hurt some latency specific programs like TELNET and Internet games but the impact is only slight. On the other hand, most HTTP and FTP traffic will SPEED UP!
- Change the MTU of all internal MASQed machines

  This solution requires the most work as you have to make minor changes to ALL of the internal IPMASQed machines. Basically, you would be changing the MTU on all of the internal machines to match the MTU of your MASQ server's Internet connection. Fortunately, this solution is usually bulletproof where as some of the other solutions mentioned in this section might rarely not work.

7.15.1. Enabling PMTU Clamping for PPPoE and some PPP Users:

For those users who use PPPoE clients for (DSL / Cablemodem) or PPP (Dialup), your Internet connection is NOT "eth0" (for example) but usually "ppp0". In addition to this, your Internet link's MTU or Maximum Transmission Unit (largest packet you can transmit over the Internet) isn't 1500 bytes but 1492. The 1492 byte MTU comes from the link size of Ethernet (1518 bytes) - Ethernet MAC overhead (18) = 1500. Then you subtract the PPPoE header (8 bytes) == MTU of 1492. This overhead isn't a big deal but sometimes ISPs or remote Internet sites do stupid things to break PPPoE or non-1500 byte MTU connected machines.

You can find more info about this topic on the web. Specifically, here is good presentaion on the topic: mss-talk presentation (PDF). Here is the entire Write up and other good info

To enable clamping in both the RP or PPPd PPPoE clients, add the following line to your /etc/ppp/pppoe.conf file:

```
# - If you have a computer acting as a gateway for a LAN, choose "1412".
#   The setting of 1412 is safe for either setup, but uses slightly more
#   CPU power.
#
CLAMPMSS=1412
```

7.15.2. Clamping the MSS via IPTABLES:

As mentioned above for PPPoE users, some ISPs and WWW sites filter critical ICMP packets like MTU Path Discovery. Because of this, many users might find more Internet sites work but others hang or work poorly. Fortunately, recent IPTABLES have added PMTU Clamping support which should help you. If your using IPTABLES and think you're hitting this issue, try adding the following line to the end of your rc.firewall-iptables ruleset. It should be noted that there is no PMTU clamping support in IPCHAINS.

```
 iptables -I FORWARD -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --clamp-mss-to-
pmtu
```

If this line causes an error when you re-run the rc.firewall-iptables* firewall rulesets, you might need to upgrade your version of IPTABLES which includes the "TCPMSS" IPTABLES module.

7.15.3. Changing the External MTU of the MASQ server:

This solution usually only applies to DIALUP users since PPPoE users cannot INCREASE their MTU because of PPPoE's header overhead.

To use this solution, first see what your current MTU for your Internet link is. To do so, run "/bin/ifconfig" on the MASQ server. Look at the lines that corresponds to your Internet connection and look for the MTU (for example, ppp0). This NEEDs to be set to 1500. Usually, Ethernet links will default to 1500 for Ethernet but serial / DIALUP modem PPP links might default to 576.

- To change the MTU on a standard Ethernet link to your bridged or routed DSL, Cablemodem, etc. connection, you need to edit the correct network startup scripts for your Linux distribution. Please see the TrinityOS - Section 16 document for network optimizations.
- To change the MTU issue on a PPP (not PPPoE) Internet link, edit your "/etc/ppp/options" file and towards the top, add the following text on two seperate lines, add:

  ```
  mtu 1500
  mru 1500
  ```

  Save these new changes and then restart PPP. Like shown above, verify that your PPP link has the correct MTU and MTU.

  CUA Users: Lastly, though this isn't a common problem, some Linux 2.0.x kernel users have found a MTU solution to the following problem. With PPP users, verify what port is your PPPd code connecting to. Is it a /dev/cua* port or a /dev/ttyS* port? It NEEDS to be a /dev/ttyS* port as the /dev/cua* device ststem has been deprecated and it breaks some things in very odd ways.

7.15.4. Changing the MTU of various operating systems:

If you reconfigure ALL of your MASQed PCs to use the SAME MTU as your external Internet link's MTU (for example, 1492 for PPPoE users), everything should work fine and this method is sometimes the MOST EFFECTIVE way of fixing things. This is including ALL of the solutions mentioned above. But doing things this way can be a lot of work if there are lots of internal MASQed machines or be even impossible to do if you don't have administrative access to all the internal MASQed machines.

Follow these simple steps for your respective operating system:

The follow examples utilize an MTU of 1492 for typical PPPoE connections for some DSL and Cablemodem users. It is recommended to use the HIGHEST values possible for all connections that are 128Kb/s and faster. It should be noted that some PPPoE ISPs might require an MTU setting of 1460 (not 1492) for proper connectivity due to additional overhead in the ISP's internal network.

The only real reason to use smaller MTUs than 1492 or 1460 is to lower your Internet link's latency but at the cost of throughput. Please see http://www.ecst.csuchico.edu/~dranch/PPP/ppp-performance.html#mtu for more details on this topic.

If you know how to make similar changes like these to other OSes like OS/2, MacOS, etc. please email David Ranch so it can be included in the HOWTO.

7.15.4.1. Changing the MTU on Linux:

```
----------------------------------------
1. The setting of MTU can vary from Linux distribution to distribution.

   For Redhat: You need to edit the various "ifconfig" statements in
               the /sbin/ifup script

   For Slackware: You need to edit the various "ifconfig" statements in
                  the /etc/rc.d/rc1.inet

2. Here is one good, any-distribution-will-work example, edit the
   /etc/rc.d/rc.local file and put the following at the END of the file:

        echo "Changing the MTU of ETH0"
        /sbin/ifconfig eth0 mtu 1492

     Replace "eth0" with the interface name that is the machine's upstream
     connection which is connected to the Internet.

3. For advanced options like "TCP Receive Windows" and such, detailed examples
   on how to edit the respective networking scripts for your specific Linux
   distro, etc., please see Chapter 16 of
   http://www.ecst.csuchico.edu/~dranch/LINUX/index-linux.html#trinityos
----------------------------------------
```

7.15.4.2. Changing the MTU on MS Windows 2000

```
----------------------------------------
1. Making ANY changes to the Registry is inheritantly risky but
   with a backup copy, you should be safe.  Proceed at your
   OWN RISK.

2. Goto Start-->Run-->RegEdit

3. Registry-->Export Registry File-->Save a copy of your registry
   to a reliable place

4. Navigate down to the key:

   [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Inter
```

```
faces\<ID for Adapter>

   Each ID Adapter has default keys for DNS, TCP/IP address, Default Gateway,
   subnet mask, etc. Find the key one that is for your network card.

5. Create the following Entry:

      type=DWORD
      name="MTU"                                  (Do NOT include the quotes)
      value=1492 (Decimal)      (Do NOT include the text "(Decimal)")

http://support.microsoft.com/support/kb/articles/Q120/6/42.asp?LN=EN-US&SD=gn&FR=0


 *** If you know how to also change the MSS, TCP Window Size, and the
 *** TTL parameters in NT 2000, please email dranch@trinnet.net as I
 *** would love to add it to the HOWTO.

5. Reboot to let the changes take effect.
-----------------------------------------
```

7.15.4.3. Changing the MTU on MS Windows NT 4.x

```
-----------------------------------------
1. Making ANY changes to the Registry is inheritantly risky but
   with a backup copy, you should be safe.  Proceed at your
   OWN RISK.

2. Goto Start-->Run-->RegEdit

3. Registry-->Export Registry File-->Save a copy of your registry
   to a reliable place

4. Create the following keys in the Registry trees, choose two
   possible Registry trees.  Multiple entries are for various
   network devices like DialUp Networking (ppp), Ethernet NICs,
   PPTP VPNs, etc.

   http://support.microsoft.com/support/kb/articles/Q102/9/73.asp?LN=EN-US&SD=gn&FR=0


   [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Parameters\Tcpip]
                     and
   [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\<Adapter-name>\Parameters
\Tcpip]

      Replace "<Adapter-Name>" with the respective name of your uplink LAN NIC
      interface

         type=DWORD
         name="MTU"              (Do NOT include the quotes)
```

```
         value=1492 (Decimal)    (Do NOT include the text "(Decimal>")

       (Do NOT include the quotes)


 *** If you know how to also change the MSS, TCP Window Size, and the
 *** TTL parameters in NT 4.x, please email dranch@trinnet.net as I
 *** would love to add it to the HOWTO.

5. Reboot to make the changes take effect.
-----------------------------------------
```

7.15.4.4. Changing the MTU on MS Windows 98:

```
-----------------------------------------
1. Making ANY changes to the Registry is inheritantly risky but
   with a backup copy, you should be safe.  Proceed at your OWN RISK.

2. Goto Start-->Run-->RegEdit

3. You should make a backup copy of your Registry before doing anything.  To
   do this, copy the "user.dat" and "system.dat" files from the \WINDOWS
   directory and put them into a safe place.  It should be noted that the
   previously mentioned method of using "Regedit: Registry-->Export Registry
   File-->Save a copy of your registry" would only perform Registry MERGES
   and NOT do a replacement.

4. Search though each of the Registry trees that end in "n" (e.g. 0007)
   and have a Registry entry called "IPAddress" which has the IP address
   of your NIC.  Under that key, add the following:

   From http://support.microsoft.com/support/kb/articles/q158/4/74.asp

     [Hkey_Local_Machine\System\CurrentControlset\Services\Class\NetTrans\000n]
         type=STRING
         name="MaxMTU"            (Do NOT include the quotes)
         value=1492 (Decimal)     (Do NOT include the text "(Decimal)")


5. You can also change the "TCP Receive Window" which sometimes
   increases network performance SUBSTANTIALLY.  If you notice your
   throughput has DECREASED, put these items BACK to their original
   settings and reboot.

     [HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\MSTCP]

        type=STRING
        name="DefaultRcvWindow"    (Do NOT include the quotes)
        value=32768 (Decimal)      (Do NOT include the text "(Decimal>")

        type=STRING
```

```
        name="DefaultTTL"          (Do NOT include the quotes)
        value=128 (Decimal)        (Do NOT include the text "(Decimal>")


6. Reboot to let the changes take effect.
-----------------------------------------
```

7.15.4.5. Changing the MTU on MS Windows 95:

```
-----------------------------------------
1. Making ANY changes to the Registry is inheritantly risky but
   with a backup copy, you should be safe.  Proceed at your OWN RISK.

2. Goto Start-->Run-->RegEdit

3. You should make a backup copy of your Registry before continuing.  To
   do this, copy the "user.dat" and "system.dat" files from the \WINDOWS
   directory and put them into a safe place.  It should be noted that the
   previously mentioned method of using "Regedit: Registry-->Export Registry
   File-->Save a copy of your registry" would only do Registry MERGES and NOT
   do a replacement.

4. Search through each of the Registry trees that end in "n" (e.g. 0007)
   and have a Registry entry called "IPAddress", which has the IP address
   of your NIC.  Under that key, add the following:

   From http://support.microsoft.com/support/kb/articles/q158/4/74.asp

     [Hkey_Local_Machine\System\CurrentControlset\Services\Class\NetTrans\000n]

         type=DWORD
         name="MaxMTU"           (Do NOT include the quotes)
         value=1492 (Decimal)    (Do NOT include the text "(Decimal)")

         type=DWORD
         name="MaxMSS"           (Do NOT include the quotes)
         value=1450 (Decimal)    (Do NOT include the text "(Decimal>")


5. You can also change the "TCP Receive Window" which sometimes
   increases network performance SUBSTANTIALLY.  If you notice your
   throughput has DECREASED, put these items BACK to their original
   settings and reboot.

     [HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\VxD\MSTCP]
        type=DWORD
        name="DefaultRcvWindow"   (Do NOT include the quotes)
        value=32768 (Decimal)     (Do NOT include the text "(Decimal>")

        type=DWORD
        name="DefaultTTL"         (Do NOT include the quotes)
```

```
        value=128 (Decimal)       (Do NOT include the text "(Decimal>")


6. Reboot to let the changes take effect.
-----------------------------------------
```

7.16. (FTP) - MASQed FTP clients don't work.

Check to see that the "ip_masq_ftp" module is loaded. To do this, log into the MASQ server and run the command "/sbin/lsmod". If you don't see the "ip_masq_ftp" module loaded, make sure that you followed the BASIC /etc/rc.d/rc.firewall-* recommendations found in Section 3.4. If you are implementing your own ruleset, make sure you include most of the examples from the HOWTO or you will have many susequent problems.

7.17. (Performance) - IP Masquerading seems slow

There might be a few reasons for this:

- You might be unrealistic about how much available bandwidth is on your modem line. Lets do the math for a typical 56k modem connection:

    1. 56k modems = 56,000 bits per second.
    2. You really DON'T have a 56k modem but a 52k modem per US FCC limitations.
    3. You'll almost NEVER get 52k, the best connection I used to get was 48k
    4. 48,000 bits per second is 4,800 BYTES per second (8 bits to a byte + 2 bits for the START and STOP RS-232 serial bits)
    5. With an MTU of 1500, you will get (3.2) packets in one second. Since this will involve fragmentation, you need to round DOWN to (3) packets per second.
    6. Again with MTU of 1500, thats 3.2 x 40 bytes of TCP/IP overhead (8%)
    7. So the BEST throughput you could hope for is 4.68KB/s w/o compression. Compression, be it v.42bis hardware compression, MNP5, or MS/Stac compression can yeild impressive numbers on highly compressable stuff like TEXT files, but acutally slow things down when transfering pre-compressed files like ZIPs, MP3s, etc.

Ethernet attached setups (DSL, Cablemodem, LANs, etc)

- Make sure you don't have both your INTERNAL and EXTERNAL networks running on the same network card with the "IP Alias" feature. If you **ARE** doing this, it can be made to work but it will be excessively slow due to high levels of collisions, IRQ usage, etc. It is highly recommended that you install another network card for the internal and external networks to have their own interface.
- Make sure you have the right Ethernet settings for both SPEED and DUPLEX.
- Some 10Mb/s Ethernet cards and most 100Mb/s cards support FULL Duplex connections. Direct connections from an Ethernet card to, say, a DSL modem (without any hubs in between) *CAN* be set to FULL DUPLEX but only if the DSL modem supports it. You should also be sure that you have Ethernet cables with all eight wires used and that they are in good condition.

  Internal networks that use HUBs -cannot- use Full Duplex. You need either a 10 or 100Mb.s Ethernet **SWITCH** to be able to do this.

Both auto 10/100Mb/s SPEED negotiation and Full/Half DUPLEX negotiation on Ethernet cards can wreck havoc on networks. I recommend to hard code both the NIC speed and duplex into the NIC(s) if possible. This is directly possible via Linux NIC kernel modules but isn't directly possible in monolithic kernels. You will need to either use MII utililies from Section 8.1 or hardcode the kernel source.

Optimize your MTU and set the TCP Sliding window to at least 8192

- Though this is COMPLETELY out of the scope of this document, this helps QUITE A BIT with ANY network link you have, be it an internal or external PPP, Ethernet, TokenRing, etc. link. For more details, this topic is briefly touched in an above section in Section 7.15. For even more details, check out the Network Optimization section of TrinityOS - Section 16.

Serial based modem users with PPP

- If you have an external modem, make sure you have a good serial cable. Also, many PCs have cheesy ribbon cables connecting the serial port from the motherboard or I/O card to the serial port connection. If you have one of these, make sure it is in good condition. Personally, I have ferrite coils (those grey-black metal like rings) around ALL of my ribbon cables.
- Make sure your MTU is set to 1500 as described in the FAQ section of this HOWTO above
- Make sure that your serial port is a 16550A or better UART. Run "dmesg | more" to verify
- Setup IRQ-Tune for your serial ports.

  On most PC hardware, the use of Craig Estey's IRQTUNE tool and significantly increase serial port performance including SLIP and PPP connections.
- Make sure that your serial port for your PPP connection is running at 115200 (or faster if both your modem and serial port can handle it.. a.k.a ISDN terminal adapters)

  - 2.0.x kernels: The 2.0.x kernels are kind of an odd ball because you can't directly tell the kernel to clock the serial ports at 115200. So, in one of your startup scripts like the /etc/rc.d/rc.local or /etc/rc.d/rc.serial file, execute the following commands for a modem on COM2:
  - setserial /dev/ttyS1 spd_vhi
  - In your PPPd script, edit the actual pppd execution line to include the speed "38400" per the pppd man page.
  - 2.2.x kernels: Unlike the 2.0.x kernels, both the 2.1.x and 2.2.x kernels don't have this "spd_vhi" issue.

    So, in your PPPd script, edit the actual pppd execution line to include the speed "115200" per the pppd man page.

All interface types:

7.18. (PORTFW) - IP Masquerading with PORTFWing seems to break when my line is idle for long periods

If you have a DSL or Cablemodem, this behavior is unfortunately quite common. Basically your ISP is putting your connection into a very low priority queue to better service non-idle connections. The problem is that some enduser's connections will actually be taken OFFLINE until some traffic from the user's DSL/Cablemodem connection awakens the ISP's hardware.

- Some DSL installations can take an idle connection OFFLINE and only be checked for activity once every 30 seconds or so.
- Some Cablemodem setups can set an idle connection into a low priority queue and only be checked for activity every minute or so.

What do I recommend to do? Ping your default gateway every 30 seconds. To do this, edit the /etc/rc.d/rc.local file and add the following to the bottom of the file:

```
ping -i 30 100.200.212.121 > /dev/null &
```

Replace the 100.200.212.121 with your default router (upstream router).

7.19. (PORTFW - Locally) - I can't reach my PORTFWed server from the INTERNAL lan

Basically, say your domain, acme.com, has an external IP address of 1.2.3.4 and you are PORTFWing all WWW traffic to an internal machine, say, 192.168.0.20/24. Then an /internal/ user on the 192.169.0.x network tries to contact to http://www.acme.com and expects things to work. Well, that isn't going to happen with a basic config. Let me explain. Basically, http://www.acme.com is being resolved to the IP of http://1.2.3.4 by your chosen DNS server. What really should be happening is the web request should resolve that request to http://192.168.0.20.

See the difference?

The proper solution to this is to setup a SPLIT DNS server. Internal users would be configured to use an /internal/ DNS server which would resolve requests like this with the 192.168.0.20 address when asked for www.acme.com. All external users should be serviced by an /external/ DNS server which will will resolve the requrest to the 1.2.3.4 IP address. From there, IPTABLES/IPCHAINS/ IPFWADM would then PORTFW the traffic to the 192.168.0.20 server as normal.

But you're probably thinking that you don't want to setup a split DNS server and there has to be another way. There are a few alternatives! The first alternative is if you only have a few internal machines. Here, you can setup a "hosts" file entry on *all* internal machines. That static hosts entry would basically look like:

```
www.acme.com    192.168.0.20
```

Got it? With that in place, the machine would consult the hosts table before going to the DNS server to resolve the host. This works well but if you change the IP address on that internal web server, you're going to need to manually update the hosts file on all of those internal machines. If you are interested in doing the more scalable split DNS approach, TrinityOS completely covers split and chrooted DNS servers. TrinityOS - Section 24 http://www.ecst.csuchico.edu/~dranch/LINUX/index-linux.html#trinityos

Now, if neither the split DNS nor the hosts file approach interests you, you still have a simple but effective alternative to get things working. What you can do is add some specific rules to your rc.firewall-* ruleset. Please see the "PORTFW Redirection of Internal requests" section under the Section 6.7 chapter.

Why didn't I mention this alternative solution first? The main reason is that it's not the ideal solution. The primary problem with this approach is that every packet will be going from the internal MASQed client to the MASQ server. There, the MASQ server will SNAT each packet to the internal MASQed WWW server's IP and then forward it to the internal web server. Once the packet is received and responded to by the web server, that response has to go back through all that processing back to the original client machine. This process is overly wasteful on both network bandwidth and MASQ server CPU cycles!

7.20. (Logs) - Now that I have IP Masquerading up, I'm getting

all sorts of weird notices and errors in the SYSLOG log files. How do I read the IPTABLES/IPCHAINS/IPFWADM firewall errors?

There is probably a few common things that you are going to see:

- **MASQ: Failed TCP Checksum error:** You might see this error when a packet coming from the Internet gets corrupt in the data section of the packet but the rest of it "seems" ok. When the Linux box receives this packet, it will calculate the CRC of the packet and determine that its corrupt. On most machines running OSes like Microsoft Windows, they just silently drop the packets but Linux IP MASQ reports it. If you get a LOT of them over your PPP link, first follow the FAQ entry above for "(Performance) - Masq seems is slow".

  If the (Performance) FAQ tips don't help and you run PPP over dialup or PPPoE, you might try adding the line "-vj" (disabled VanJacobson header compression) to your /etc/ppp/options file and restart the PPPd connection.
- **Firewall hits**: Because you are on the Internet with a decent firewall, you will be surprised with the number of users trying to penetrate your Linux box! So what do all these firewall logs mean?

  More so, if they are filling your logs, see the next FAQ entry on thoughts *how to reduce* all these log entries.
- The following details are from the TrinityOS - Section 10 documentation I also wrote:

```
With the use of various firewall rulesets, a given ruleset can either
DENY (silently drop) or REJECT traffic (sends back a ICMP error).  If
firewall logging is enabled, the errors will show up in the SYSLOG
"messages" file found at:

    Redhat:         /var/log
    Slackware:      /var/adm

If you take a look at one of these firewall logs, you would see something
like:

---------------------------------------------------------------------
IPTABLES:
---------
Feb 23 07:37:01 Roadrunner kernel: IPTABLES IN=eth0 OUT=
  MAC=00:50:da:2e:e5:fb:00:03:47:73:c9:d2:08:00 SRC=12.75.147.174
  DST=100.200.0.212 LEN=44 TOS=0x00 PREC=0x00 TTL=64 ID=39034 DF PROTO=TCP
  SPT=4313 DPT=23 WINDOW=32120 RES=0x00 SYN URGP=0


IPCHAINS:
---------
Feb 23 07:37:01 Roadrunner kernel: input REJECT eth0 PROTO=6
  12.75.147.174:1633 100.200.0.212:23 L=44 S=0x00 I=54054 F=0x0040 T=64


IPFWADM:
--------
Feb 23 07:37:01 Roadrunner kernel: IP fw-in rej eth0 TCP 12.75.147.174:1633
  100.200.0.212:23 L=44 S=0x00 I=54054 F=0x0040 T=64

---------------------------------------------------------------------
```

```
There is a LOT of information in just this one line of SYSLOG.  Lets break
out this example.  You should refer back to the original firewall hit as you
read this.

--------------

    1. ================================================================

        - This packet firewall "hit" occurred on "Feb 23 07:37:01"

    2. ================================================================

        - This packet was logged on the "RoadRunner" computer via the kernel

    3. ================================================================

        - IPTABLES: the SYSLOG prepend string is "iptables" for information purposes

    - IPCHAINS: the packet was stopped on the INPUT chain

    - IPFWADM:  the packet was an IP packet

    4. ================================================================

    - IPTABLES: the packet came IN on interface "eth0"

    - IPCHAINS: the packet was REJECTED (vs. dropped or accepted)

    - IPFWADM:  the packet was stopped on INPUT (vs. "fw-out" for OUT or
       "fw-fwd" for FORWARD)

    5. ================================================================

    - IPTABLES: the packet had NO output interface

    - IPCHAINS: the packet came in on the "eth0" interface

    - IPFWADM:  the packet was REJECTED "rej" (vs. "deny" or "accept")

    6. ================================================================

    - IPTABLES: this display's the MAC address of the source and destination
                Ethetnet MAC address (only relivant for Ethernet networks)

    - IPCHAINS: the packet was IP protocol 6 or TCP
                        * If you don't know that protocol 6 is for TCP, look at
                  your /etc/protocols file to see what other protocol numbers
                  are used for.

    - IPFWADM:  the packet on the "eth0" interface

    7. ================================================================

    - IPTABLES: the packet's source IP address was 12.75.147.174
```

```
    - IPCHAINS: the packet's source IP address was 12.75.147.174

- IPFWADM:  the packet was a "TCP" packet

8. ================================================================

    - IPTABLES: the packet's destination IP address was 100.200.0.212

- IPCHAINS: the packet's source PORT was 1633

- IPFWADM:  the packet's source IP address was 12.75.147.174

9. ================================================================

- IPTABLES: the packet's length was 44 bytes

    - IPCHAINS: the packet's destination IP address was 100.200.0.212

- IPFWADM:  the packet's source PORT was 1633

10. ===============================================================

- IPTABLES: the packet's TOS markings (type of service which basically
           means class of service) was 0x00 or zero.

- IPCHAINS: the packet's destination PORT was 23 (telnet)
                  * If you don't know that port 23 is for TELNETing, look at
              your /etc/services file to see what other ports are used
              for.

- IPFWADM:  the packet's destination IP address was 100.200.0.212

11. ===============================================================

- IPTABLES: the packet's precedense markings (class of service) was
           0x00 or zero.

- IPCHAINS: the packet's length was 44 bytes

- IPFWADM:  the packet's destination PORT was 23 (telnet)
                  * If you don't know that port 23 is for TELNETing, look at
              your /etc/services file to see what other ports are used
              for.

12. ===============================================================

- IPTABLES: the packet's TTL or Time to Live was 64 or 64 router hops
                  * Every router hop over the Internet will subtract (1) from
              this number.  Usually, packets will start with a number of
              255 (depends on the operating system) and if that number
              ever reaches (0), it means that realistically, the packet
              was lost in a network loop and should be deleted.

- IPCHAINS: the packet's TOS markings (type of service which basically
```

```
                means class of service) was 0x00 or zero.
                        * divide this by 4 to get the Type of Service (presidence)

    - IPFWADM:  the packet was 44 bytes long

    13. ================================================================

        - IPTABLES: the packet had various TCP flags set such as SYN, SYN+ACK,
                etc. (shown in HEX)

    - IPCHAINS: the packet had various TCP flags set (shown in hex)

    - IPFWADM:  the packet's TOS markings (type of service which basically
                means class of service) was 0x00 or zero.

    14. ================================================================

        - IPTABLES: the packet's "don't fragment" or DF bit was set from the
                source computer

    - IPCHAINS: the packet had a fragmentation offset of 40 (shown in HEX)

                        --Don't worry if you don't understand this..
                        * A value that started with "0x2..." or "0x3..." means the
                  "More Fragments" bit was set so more fragmented packets
                  will be coming in to complete this one BIG packet.
                        * A value which started with "0x4..." or "0x5..." means
that
                  the "Don't Fragment" bit was set
                        * Any other values are the Fragment offset (divided by 8)
to
                  be later used to recombine into the original LARGE packet

        - IPFWADM:  the packet had various TCP flags set such as SYN, SYN+ACK,
                etc. (shown in HEX)

    15. ================================================================

        - IPTABLES: the packet was a TCP packet

    - IPCHAINS: the packet's TTL or Time to Live was 64 or 64 router hops
                        * Every router hop over the Internet will subtract (1) from
                  this number.  Usually, packets will start with a number of
                  255 (depends on the operating system) and if that number
                  ever reaches (0), it means that realistically, the packet
                  was lost in a network loop and should be deleted.

    - IPFWADM:  the packet had a fragmentation offset of 40 (shown in HEX)

                        --Don't worry if you don't understand this..
                        * A value that started with "0x2..." or "0x3..." means the
                  "More Fragments" bit was set so more fragmented packets
                  will be coming in to complete this one BIG packet.
                        * A value which started with "0x4..." or "0x5..." means
that
```

```
                  the "Don't Fragment" bit was set
                        * Any other values are the Fragment offset (divided by 8)
to
                  be later used to recombine into the original LARGE packet

    16. ============================================================

    - IPTABLES: the packet's soure PORT was 4313

    - IPCHAINS:

    - IPFWADM:  the packet's TTL or Time to Live was 64 or 64 router hops
                        * Every router hop over the Internet will subtract (1) from
                  this number.  Usually, packets will start with a number of
                  255 (depends on the operating system) and if that number
                  ever reaches (0), it means that realistically, the packet
                  was lost in a network loop and should be deleted.

    17. ============================================================

    - IPTABLES: the packet's destination PORT was 23 (telnet)
                        * If you don't know that port 23 is for TELNETing, look at
                  your /etc/services file to see what other ports are used
                  for.

    - IPCHAINS:

    - IPFWADM:

    18. ============================================================

    - IPTABLES: the packet's TCP window (sliding or selective TCP ack)
                was 32120 bytes

    - IPCHAINS:

    - IPFWADM:

    19. ============================================================

    - IPTABLES: the packet's TCP reserved bits were 0x00 (HEX) - unused

    - IPCHAINS:

    - IPFWADM:

    20. ============================================================

    - IPTABLES: the packet's TCP header SYN bit was set
                * IPTABLES displays all the TCP header bits by name and not
                  by a HEX dump

    - IPCHAINS:

    - IPFWADM:
```

```
21. ================================================================

- IPTABLES: the packet's TCP header URGENT bit was set - rarely used

- IPCHAINS:

- IPFWADM:
```

7.21. (Log Reduction) - My logs are filling up with packet hits due to the new "stronger" rulesets. How can I fix this?

So your realizing that a good firewall is catching a LOT of bad Internet traffic. That's a good thing but it's also filling up your logs to the point that you won't read them; that's bad. What to do?

What you need to figure out is what traffic you DON"T want to log, explicitly match those packets in the firewall, and NOT log the packets when you drop them.

For example, the TrinityOS firewall ruleset in section 10.7 (this would be a "strongest" ruleset in IPMASQ speak) gives some ideas: TrinityOS - Section 10.7

Things I recommend to filter:

- All RFC1918 address space (TCP/IP address ranges: 10.x.y.z/8, 172.16-31.y.z/12, and 192.168.y.x/16). You should /never/ receive these packets from an Internet connection. If you do, they are most likely spoofed packets
- Windows File and Print Sharing (Samba or CIFS): ports 137, 138, 139, and 445. Windows machines like to talk a lot though most computers don't care what they're saying.
- Class-D Multicast addresses (if you don't use Multicast): 224.0.0.0/4
- Class-E and F "future" addresses: 240.0.0.0/5 and 248.0.0.0/5

To a much lesser extent, you might want to filter other packets. I recommend that you verify that you are receiving these specific packet types before you filter them out.

- RIP (the routing protocol): port 520
- Some specific forms of ICMP packets - NOT all of them (that will break your machine and IPMASQ in general)

Finally, you'll probably find that some individual TCP/IP address out on the Internet always seem to attack your IP. So, in addition to filtering various PORTS like above, you might want to also filter by specific SOURCE IP address too. After all, it is *YOUR* firewall.

7.22. (MASQ Security) - Can I configure IP MASQ to allow Internet users to directly contact internal MASQed servers?

Yes! With IPPORTFW, you can allow ALL or only a select few Internet hosts to contact ANY of your internal MASQed computers. **This topic is completely covered in Section 6.7 in this HOWTO.**

7.23. (Free Ports) - I'm getting "kernel: ip_masq_new (proto=UDP): no free ports." in my SYSLOG files. Whats up?

One of your internal MASQed machines are creating an abnormally high number of packets destined for the Internet. As the IP Masq server builds the MASQ table and forwards these packets out over the Internet, the table is quickly filling. Once the table is filled, it will give you this error.

The only application that I have known which temporarily creates this situation is a gaming program called "GameSpy". Why? Gamespy builds a server list and then pings all of the servers in the list (1000s of game servers). By creating all these pings, it creates 1,000s of quick connections in a VERY short period of time. Until these sessions timeout via the IP MASQ timeouts, the MASQ tables become "FULL".

So what can you do about it? Realistically, don't use programs that do things like this. If you do get this error in your logs, find it and stop using it. If you really like GameSpy, just don't refresh the server too often. Regardless, once you stop running this MASQ'ed program, this MASQ error will go away as these connections will eventually timeout in the MASQ tables.

7.24. (SETSOCKOPT) - I'm getting "ipfwadm: setsockopt failed: Protocol not available" when I try to use IPPORTFW!

If you get the error message "ipfwadm: setsockopt failed: Protocol not available", you AREN'T running your new kernel. Make sure that you moved the new kernel over, re-run LILO, and then reboot again.

Please see the end of Section 6.7 for full details.

7.25. (SAMBA) - Microsoft File and Print Sharing and Microsoft Domain clients don't work through IP Masq!

To properly support Microsoft's SMB protocol, an IP Masq module would need to be written but there are three viable work-arounds. For more details, please see this Microsoft KnowledgeBase article.

The first way to work around this problem is to configure IPPORTFW from Section 6.7 and portfw TCP ports 137, 138, and 139 to the internal Windows machine's IP address. Though this solution works, it would only work for ONE internal machine.

The second solution is to install and configure Samba on the Linux MASQ server. With Samba running, you can then map your internal Windows File and Print shares onto the Samba server. Then, you can mount these newly mounted SMB shares to all of your external clients. Configuring Samba is fully covered in a HOWTO found in a Linux Documentation Project and in the TrinityOS document as well.

The third solution is to configure a VPN (virtual private network) between the two Windows machines or between the two networks. This can either be done via the PPTP or IPSEC VPN solutions. There is a Section 7.35 patch for Linux and also a full IPSEC implementation available for both 2.0.x and 2.2.x kernels. This solution would probably be the most reliable and secure method of all

three solutions.

All of these solutions are NOT covered by this HOWTO. I recommend that you look at the TrinityOS documentation for IPSEC help and John Hardin's PPTP page for more information.

Also PLEASE understand that Microsoft's SMB protocol is VERY insecure. Because of this, running either Microsoft File and Print sharing or Windows Domain login traffic over the Internet without any encryption is a VERY BAD idea.

7.26. (IDENT) - IRC won't work properly for MASQed IRC users. Why?

The main possible reason is because most common Linux distribution's IDENT or "Identity" servers can't deal with IP Masqueraded links. No worries though, there are IDENTs out there that will work.

Installing this software is beyond the scope of this HOWTO but each tool has its own documentation. Here are some of the URLs:

- Oident is a favorite IDENT server for MASQ users.
- Mident is another popular IDENT server.
- Sident
- Other Idents

Please note that some Internet IRC servers still won't allow multiple connections from the same host even if they get Ident info and the users are different though. Complain to the remote sys admin. :-)

7.27. (IRC DCC) - mIRC doesn't work with DCC Sends

This is a configuration problem on your copy of mIRC. To fix this, first disconnect mIRC from the IRC server. Now in mIRC, go to File --> Setup and click on the "IRC servers tab". Make sure that it is set to port 6667. If you require other ports, see below. Next, goto File --> Setup --> Local Info and clear the fields for Local Host and IP Addresses. Now select the checkboxes for "LOCAL HOST" and "IP address" (IP address may be checked but disabled). Next under "Lookup Method", configure it for "normal". It will NOT work if "server" is selected. That's it. Try to the IRC server again.

If you require IRC server ports other than 6667, (for example, 6969) you need to edit the /etc/rc.d/rc.firewall-* startup file where you load the IRC MASQ modules. Edit this file and the line for "modprobe ip_masq_irc" and add to this line "ports=6667,6969". You can add additional ports as long as they are separated with commas.

Finally, close down any IRC clients on any MASQed machines and re-load the IRC MASQ module:

/sbin/rmmod ip_masq_irc /etc/rc.d/rc.firewall-*

7.28. (IP Aliasing) - Can IP Masquerade work with only ONE Ethernet network card?

Yes and no. With the "IP Alias" kernel feature, users can setup multiple aliased interfaces such as eth0:1, eth0:2, etc but its is NOT recommended to use aliased interfaces for IP Masquerading. Why? Providing a secure firewall becomes very difficult with a single NIC card. In addition to this, you will experience an abnormal amount of errors on this link since incoming packets will almost simultaneously be sent out at the same time. Because of all this and NIC cards now costs less than $10, I highly recommend to just get a NIC card for each MASQed network segment.

Users should also understand that IP Masquerading will only work with a physical interface such as eth0, eth1, etc. MASQing out an aliased interface such as "eth0:1, eth1:1, etc" will NOT work. In other words, the following WILL NOT WORK reliably:

- It is rumored that you can simply use the destination IP address (the IP address associated with the ALIASed interface like eth0:1, etc.) IN PLACE of specifing the interface (eth0:1). This solution is not untested -- please email dranch@trinnet.net if you have any positive or negative results
- /sbin/ipchains -A forward -i eth0:1 -s 192.168.0.0/24 -j MASQ"
- /sbin/ipfwadm -F -a m -W eth0:1 -S 192.168.0.0/24 -D 0.0.0.0/0

If you are still interested in using aliased interfaces, you need to enable the "IP Alias" feature in the kernel. You will then need to re-compile and reboot. Once running the new kernel, you need to configure Linux to use the new interface (i.e. eth0:1, etc.). After that, you can treat it as a normal Ethernet interface with some restrictions like the one above.

7.29. (Multiple-LANs) - I have two MASQed LANs but they cannot communicate with each other!

Please see Section 6.5 for full details.

7.30. (SHAPING) - I want to be able to limit the speed of specific types of traffic

I receive lots of emails from people asking something like the following:

```
How can I control the internet bandwidth among the LAN PCs behind the IPMASQ
server? Some times any local pc is downloading and it it will take the majority
of the bandwidth and thus the other PCs get little bandwidth.
```

This topic really doesn't have anything to do with IPMASQ and everthing to do with Linux's built-in traffic shaping and rate-limiting abilities. You will find information about this on the LDP such as the ADSL Bandwidth HOWTO, the Advanced Rouring HOWTO, and the Bandwidth Limiting HOWTO. Please understand this is an advanced topic and any question you may have will be better served from people from those forums.

7.31. (ACCOUNTING) - I need to do accounting on who is using the network

Though this doesn't have much to do with IPMASQ, here are a few ideas. If you know of a better solution, please email the author of this HOWTO so it can be added to the HOWTO.

- Idea #1: You could run the command:

  ```
  IPCHAINS: "ipchains -L -M"

  IPTABLES: "cat /proc/net/ip_conntrack"

  IPFWADM:
  ```

  once a second and log all of those entries. You could then write a program to merge this information into one large file. Again, this will only provide you with the remote IP address and nothing about the content viewed or downloaded.

- Idea #2: Log every packet: You can match any specific traffic flows but this method will create VERY LARGE log files. Unfortunately, these log files aren't very readable and it doesn't tell you what was transfered (FTP files, etc.). Fortunately, setting up this form of accounting is easy.
- Idea #3: Say you want to log all traffic going out onto the internet. You can setup a firewall rule to accept port 80 traffic with with the SYN bit set and log it. Now mind you, this will create smaller log files than the idea above but you will only know the destination IP address and NOT the WWW pages viewed.
- Idea #4: Transparent Proxy: This method really doesn't use IPMASQ since it requires the installation and setup of the Squid HTTP/FTP proxy server. The benefit of this method is that internal users won't notice anything different in terms of connectivity but now the SysAdmin gets a LOT more information (files downloaded, etc). But, there are pros/cons to setting this up:

  Pro:

  - + full logging of all transferred files and issues FTP commands
  - + you can enable caching on the proxy server. With caching, you can save bandwidth since once a file is downloaded, any identical file requests will be served via the cache and not redownloaded via the Internet connection.

  Con:

  - - Setting up a transparent proxy is complicated as it requires kernel changes, setting up Squid, etc.
  - - Could be overkill for a small installation.

  Please see the Advanced Routing HOWTO for more details.

7.32. (MULTIPLE IPs - DMZ segments) - I have several EXTERNAL IP addresses that I want to PORTFW to several internal machines. How do I do this?

Though technically possible, DON'T do this with IP MASQ. There are far better solutions for this network design.

MASQ is a 1:Many NAT setup which is the incorrect tool to perform what you are looking for. You are looking for is either Many: Many NAT solution or a Briding setup.

NOTE: For users out there who are thinking about enabling multiple IP addresses on one internal NIC using "IP Alias" and then just

PORTFWeding ALL of those ports (0-65535), and and finally use IPROUTE2 to maintain the proper source/destination IP pairs. This has been done SUCCESSFULLY on 2.0.x kernels and less successfully on 2.2.x kernels. Regardless of success, that isn't the proper way to do it, it's a total HACK, and it is not a supported MASQ configuration. Please, give IPTABLES on the 2.4.x kernels a serious look or to a much lesser extent, Section 7.30 IPROUTE2 look for 2.2.x kernels.

Anyway, for forwarding external IP address to internal hosts, you basically have three possibilites:

-

```
1. Route the external IPs

   (This does NOT involve IPMASQ at all but requires special WAN
addressing
    and routing setup from your ISP):

    Internet -- Some public WAN -- Linux -- DMZ segment
                  IP address      Server     PUBLIC IPs
                                    |
                                    +------ Internal net
                                             private IPs
```

-

```
2. 1:1 NAT

   (Most easily done via IPTABLES or with IPCHAINS and IPROUTE2 but
still
    some protocols cannot deal with NAT)

    Internet -- Linux -- DMZ segment
                Server     Private IPs natted to 1:1 PUBLIC IPs
                   |
                   +------ Internal net
                            private IPs
```

-

```
3. Bridging or ProxyARP:

   The Bridging method is one of the more popular methods that many
commercial
   firewalls do and it's very slick.  Alternatively, you can use the ProxyARP
   method which works well without some of the complications (or benefits of
   bridging).  With both solutions, all public IPs can transparently flow
   through the Linux server to the DMZ but via firewall inspection.

    Internet -- Linux -- DMZ segment
                Server     PUBLIC IPs
                  |
                  +------ Internal net
                           private IPs
```

Each of these solutions have pros and cons

Item #1: If you're lucky enough to have an ISP that will set this up for you (pretty rare), all you need to do is use basic 'route' commands to get this running. This is the most rebust solution and doesn't require any form of IPMASQ or NAT to work.

Item #2: 1:1 NAT isn't covered in this HOWTO yet but if you need a hand, just email me and I'll give you a hand.

Item #3: ProxyARP is pretty strait forward but only works in specific situations and only works with Ethernet networks. Bridging is more powerful but will probably require the re-compiling of the kernel and some advanced configuration. Ultimately, neither of these solutions are IPMASQ anymore and thus I can't help you set them up. Fortunately, there are other HOWTOs out there that cover this topic:

- http://www.tldp.org/HOWTO/Proxy-ARP-Subnet/index.html
- http://www.tldp.org/HOWTO/Adv-Routing-HOWTO/lartc.bridging.html
- http://www.tldp.org/HOWTO/Ethernet-Bridge-netfilter-HOWTO.html
- http://bridge.sourceforge.net/

NOTE: If you have a bridged DSL or Cablemodem connection (not PPPoE), things are a little more difficult because your setup isn't routed. No worries though, check out the Bridge+Firewall Mini HOWTO and the Bridge+Firewall+DSL Mini HOWTO. These HOWTOs will teach you how to get your Linux box to support multiple IP addresses on a single interface!

7.33. (1:1 NAT) - I'd like to do 1:1 NAT but I can't figure out how to do it

Please see the previous FAQ entry named Section 7.32 for all the details.

7.34. (Netstat) - I'm trying to use the NETSTAT command to show my Masqueraded connections but its not working

There might be a problem with the "netstat" program in 2.0.x-based Linux distros. After a Linux reboot, running "netstat -M" works fine but after a MASQed computer runs some successful ICMP traffic like ping, traceroute, etc., you might see something like:

masq_info.c: Internal Error `ip_masquerade unknown type'.

The workaround for this is to use the "/sbin/ipfwadm -M -l" command. You will also notice that once the listed ICMP masquerade entries timeout, "netstat" works again.

7.35. (VPNs) - I would like to get Microsoft PPTP (GRE tunnels) and/or IPSEC (Linux SWAN) tunnels running through

IP MASQ

This IS possible for specific modes. Specifically, all of the kernel versions (2.4.x, 2.2.x, and 2.0.x) support patches to allow for both ONE or MULTIPLE PPTP users behind a IPMASQ server to connect to the -same- PPTP server. The 2.4.x kernels currently have a PPTP module now in the newest versions of the IPTABLES program and there is another version available on the IPMASQ WWW site. Please check out John Hardin's PPTP Masq page for details.

7.36. (Games) - I want to get the XYZ network game to work through IP MASQ but it won't work. Help!

First, check Steve Grevemeyer's MASQ Applications page. If your solution isn't listed there, try patching your Linux kernel with Glenn Lamb's http://ipmasq.webhop.net/files20/loose-udp-2.0.36.patch.gz LooseUDP patch which is covered in Section 6.10 above. Also check out Dan Kegel's NAT Page for more information.

If you are technically inclined, use the program "tcpdump" and sniff your network. Try to find out what protocols and port numbers your XYZ game is using. With this information in hand, subscribe to the IP Masq email list and email your results for help.

7.37. (Stops working) - IP MASQ works fine for a while but then it stops working. A reboot seems to fix this. Why?

I bet you are using IPAUTOFW and/or you have it compiled into the kernel huh?? This is a known problem with IPAUTOFW. It is recommend to NOT even configure IPAUTOFW into the Linux kernel and use IPPORTFW option instead. This is covered with more details in Section 6.7.

7.38. (SMTP Relay) - Internal MASQed computers cannot send SMTP or POP-3 mail!

Though this isn't a Masquerading issue but many users do this so it should be mentioned.

SMTP: The issue is that you are probably using your Linux box as an SMTP relay server and get the following error:

```
"error from mail server: we do not relay"
```

Newer versions of Sendmail and other Mail Transfer Agents (MTAs) disable relaying by default (this is a good thing). So do the following to fix this:

- Sendmail: Enable specific relaying for your internal MASQed machines by editing the /etc/sendmail.cw file and add the hostname and domain name of your internal MASQed machine. You should also check to see that the /etc/hosts file has the IP address and Fully Qualified Domain Name (FQDN) configured in it. Once this is done, you need to restart Sendmail for it to re-read its configuration files. This is covered in TrinityOS - Section 25

POP-3: Some users configure their internal MASQ'ed computer's POP-3 clients to connect to some external SMTP server. While this is fine, many SMTP servers out there will try to IDENT your connection on port 113. Most likely your problem stems around your default Masquerade policy being set to DENY. This is BAD. Set it to REJECT and re-run your rc.firewall-* ruleset.

7.39. (Source Routing) - I need different internal MASQed networks to exit on different external IP addresses

Say you have the following setup: You have multiple internal networks and also multiple external IP addresses and/or networks. What you want to do is have LAN #1 to only use External IP #1 but you wan LAN #2 to use External IP #2.

Internal LAN ----------> official IP

LAN #1 External IP #1 192.168.0.x --> 123.123.123.11

LAN #2 External IP #2 192.168.1.x --> 123.123.123.12

Basically, what we have described here is routing NOT only on the destination address (typical IP routing) but also routing based upon the SOURCE address as well. This is typically called "policy-based routing" or "source routing". This functionality is NOT available in 2.0.x kernels, it *IS* available for 2.2.x kernels via the IPROUTE2 package, and it is built into the new 2.4.x kernels using IPTABLES.

First, you have to understand that both IPFWADM and IPCHAINS get involved *AFTER* the routing system has decided where to send a given packet. This statement really ought to be stamped in big red letters on all IPFWADM/IPCHAINS/IPMASQ documentation. The reason for this is that users MUST first have their routing setup correct, then start adding IPFWADM/IPCHAINS and/or Masq features.

Anyways, for the example case shown above, you will need to persuade the routing system to direct packets from 192.168.0.x via 123.123.1233.11 and packets from 192.168.1.x via 123.123.123.12. That is the hardest part and adding Masq on top of correct routing is easy.

To do this fancy routing, you will use IPROUTE2. Because this functionality has NOTHING to do with IPMASQ, this HOWTO does not cover this topic in great detail. Please see Section 2.7 for complete URLs and documentation for this topic.

The "iprule" and "iproute" commands are the same as "ip rule" and "ip route" commands (I prefer the former since it is easier to search for.) All the commands below are completely untested, if they do not work, please let David Ranch know about it but please contact the IPROUTE2 email list for help. This function has NOTHING to do with IP Masquerading.

2.4.x. kernels:

The following would be integrated into the END of your rc.firewall-iptables ruleset

```
EXTIF="eth0"
INTNET1="192.168.0.0/24"
INTNET2="192.168.1.0/24"
EXTIP1="123.123.123.11"
EXTIP2="123.123.123.12"

iptables -t nat -A POSTROUTING -o $EXTIF -s $INTNET1 -j SNAT --to $EXTIP1
iptables -t nat -A POSTROUTING -o $EXTIF -s $INTNET2 -j SNAT --to $EXTIP2
```

2.2.x. kernels:

The first few commands only need to be done once at boot, say in /etc/rc.d/rc.local file.

```
# Allow internal LANs to route to each other, no masq.
  /sbin/iprule add from 192.168.0.0/16 to 192.168.0.0/16 table main pref 100
# All other traffic from 192.168.1.x is external, handle by table 101
  /sbin/iprule add from 192.168.1.0/24 to 0/0 table 101 pref 102
# All other traffic from 192.168.2.x is external, handle by table 102
  /sbin/iprule add from 192.168.2.0/24 to 0/0 table 102 pref 102

These commands need to be issued when eth0 is configured, perhaps in
/etc/sysconfig/network-scripts/ifup-post (for Redhat systems).  Be sure to
do them by hand first to make sure they work.

# Table 101 forces all assigned packets out via 123.123.123.11
  /sbin/iproute add table 101 via 123.123.123.11
# Table 102 forces all assigned packets out via 123.123.123.12
  /sbin/iproute add table 102 via 123.123.123.12

At this stage, you should find that packets from 192.168.1.x to the
outside world are being routed via 123.123.123.11, packets from
192.168.2.x are routed via 123.123.123.12.

It is IMPORTANT that these IPROUTE2 rules be run /BEFORE/ the rc.firewall-*
ruleset is run.

If everything hangs together, the masq code will see packets being
routed out on 123.123.123.11 and 123.123.123.12 and will use those addresses
as the masq source address.
```

7.40. (IPCHAINS rulesets on 2.4.x kernels) - What the ipchains.o module can do on 2.4.x kernels

Some people would like to continue using their legacy IPCHAINS rulesets on 2.4.x-based kernelw. Unfortunately, unless you are **only doing packet firewalling** and not trying to do any NATing (MASQ), PORTFW, or other advanced features, you're in trouble.

- If you ARE only doing IPCHAINS filtering, all you need to do is unload all IPTABLES modules shown from the "`/sbin/`

`lsmod`" command. After that, load the IPCHAINS module by running "`/sbin/modprobe ipchains`". After that, load your IPCHAINS ruleset as normal.

- Please note that if you compiled IPTABLES support statically into the kernel, you CANNOT load the "ipchains" module (it shouldn't be even present) as it will conflict with the IPTABLES kernel code. Your ONLY option in this case is to recompile your kernel but make the IPTABLES and IPCHAINS options as modules.

So why can't you run IPCHAINS MASQ/PORTFW functionality with a 2.4.x kernel? Once the IPCHAINS module is loaded, you CANNOT use any IPTABLES commands or modules since the code conflicts. In addition to this, you cannot use any legacy 2.2.x IPCHAINS masq modules on a 2.4.x kernel as the kernels are so radically different. Plus, this really shouldn't be an issue as all of this functionality is available via native IPTABLES modules now. Finally, you cannot use the IPMASQADM tool with a 2.4.x kernel as the program both won't compile and ultimately the PORTFW kernel handlers aren't present anymore (it's now done natively by the IPTABLES code). So, considering all of these facts:

- You cannot run any form of PORTFW on this 2.4.x machine
- Protocols that require special handling like FTP, IRC, CuSeeme, RealAudio, etc. will no longer work

Basically, the ipchains kernel module included with the 2.4.x kernels is intended for basic packet firewall compatibility and NOT any NAT(MASQ) functionality.

7.41. (IPTABLES vs. IPCHAINS vs. IPFWADM) - Why do the 2.4.x, 2.2.x, and 2.0.x kernels use different firewall systems?

IPTABLES supports the following features that IPCHAINS and IPFWADM doesn't:

- Stateful IPv4 protocol and application tracking
- Stateful IPv6 protocol tracking
- True 1:1 and 1:Many NAT
- Built-in PORTFW functionality
- See the Section 2.6 section for more details

IPCHAINS supports the following features that IPFWADM doesn't:

- "Quality of Service" (QoS support)
- A TREE style chains system vs. LINEAR system like IPFWADM (Eg. this allows something like "if it is ppp0, jump to this chain (which contains its own difference set of rules)"
- IPCHAINS is more flexible with configuration. For example, it has the "replace" command (in addition to "insert" and "add"). You can also negate rules (e.g. "discard any outbound packets that don't come from my registered IP" so that you aren't the source of spoofed attacks).
- IPCHAINS can filter any IP protocol explicitly, not just TCP, UDP, ICMP

7.42. (Upgrades) - I've just upgraded to the x.y.z kernel, why isn't IP Masquerade working?

There are several things you should check assuming your Linux IP Masq box already has proper connections to the Internet and your LAN:

- Make sure you have the necessary features and modules are compiled and loaded. See earlier sections for details.
- Check `/usr/src/linux/Documentation/Changes` and make sure you have the minimal requirement for the network tools installed.
- Make sure you followed all of the tests in Chapter 5 of the HOWTO.
- Make sure you are using the proper firewall tool for you kernel be it IPTABLES, IPCHAINS, or IPFWADM.
- If you are doing PORTFW functionality, make sure you use the correct tool for your kernel version. IPTABLES has everything built-in, IPCHAINS requires the use of IPMASQADM, and IPFWADM reaquires the use of IPPORTFW or IPAUTOFW. This is completely covered in Section 6.7.
- Go through all setup and configuration again! Usually, it's just a typo or a simple mistake you are overlooking.

7.43. (EQL) - I need help with EQL connections and IP Masq

EQL has nothing to do with IP Masq though they are commonly teamed up on Linux boxes. Because of this, I recommend checking out the NEW version of Robert Novak's EQL HOWTO for all your EQL needs.

7.44. (Wussing out) - I can't get IP Masquerade to work! What options do I have for Windows Platforms?

Looking to give up a free, reliable, high performance solution that works on minimal hardware and pay a fortune for something that needs more hardware, is lower performance and did I say less reliable? (IMHO. And yes, I have real life experience with these ;-)

Okay, it's your call. If you want a Windows NAT and/or proxy solution, here is a decent listing. I don't prefer any one of these tools over another, especially since I haven't used them before.

- Firesock (from the makers of Trumpet Winsock)
  - Does Proxy
  - http://www.trumpet.com.au
- Iproute
  - DOS program designed to run on 286+ class computers
  - requires another box like Linux MASQ
  - (UNAVAILABLE) www.mischler.com/iproute/
- Microsoft Proxy
  - Requires Windows NT Server
  - Quite expensive
  - http://www.microsoft.com
- NAT32
  - Windows 95/98/NT compatible
  - http://www.nat32.com

- Roughly $25 for Win9x and $47 for Win9x and WinNT

- SyGate
  - http://www.sygate.com

- Wingate
  - Does proxy
  - Costs roughly $30 for 2-3 IPs
  - http://www.wingate.com

- Winroute
  - Does NAT
  - WinRoute

Lastly, do a web search on "MS Proxy Server", "Wingate", "WinProxy", or goto www.download.com. And definitely DON'T tell anyone that we sent you.

7.45. (Developers) - I want to help with IP Masquerade development. What can I do?

Join the Linux IP Masquerading DEVELOPERS list and ask the developers there what you can do to help. For more details on joining the list, check out Section 7.5 FAQ section.

Please DON'T ask NON-IP-Masquerade development related questions there!!!!

7.46. (More INFO) - Where can I find more information on IP Masquerade?

You can find more information on IP Masquerade at the Linux IP Masquerade Resource that Ambrose Au maintains.

You can also find more information at Dranch's Linux page where the TrinityOS and other Linux documents are kept.

You may also find more information at The Semi-Original Linux IP Masquerading Web Site maintained by Indyramp Consulting, who also provides the IP Masq mailing list.

Lastly, you can look for specific questions in the IP MASQ and IP MASQ DEV email archives or ask a specific question on these lists. Check out Section 7.5 FAQ item for more details.

7.47. (Translators) - I want to translate this HOWTO to another

language, what should I do?

Make sure the language you want to translate to is not already covered by someone else. But, most of the translated HOWTOs are now OLD and need to be updated. A list of available HOWTO translations are available at the Linux IP Masquerade Resource.

If a copy of a **current** IP MASQ HOWTO isn't in your proposed language, please download the newest copy of the IP-MASQ HOWTO SGML code from the Linux IP Masquerade Resource. From there, begin your work while maintaining good SGML coding. For more help on SGML, check out www.sgmltools.org

---

7.48. (Updates) - This HOWTO seems out of date, are you still maintaining it? Can you include more information on ...? Are there any plans for making this better?

Yes, this HOWTO is still being maintained. In the past, Ambrose was guilty of being too busy working on two jobs and didn't have much time to work on this, my apologies. As of v1.50, David Ranch revamped the document and got it current again.

If you think of a topic that could be included in the HOWTO, please send email dranch@trinnet.net. It will be even better if you can provide that information. We will then include the information into the HOWTO once it is both found appropriate and tested. Many thanks for your contributions!

We have a lot of new ideas and plans for improving the HOWTO, such as case studies that will cover different network setup involving IP Masquerade, more on security via strong IPFWADM/IPCHAINS firewall rulesets, IPCHAINS usage, more FAQ entries, etc. If you think you can help, please do! Thanks.

7.49. (Thanks) - I got IP Masquerade working, it's great! I want to thank you guys, what can I do?

- Can you translate the newer version of the HOWTO to another language?
- Thank the developers and appreciate the time and effort they spent on this.
- Join the IP Masquerade email list and support new MASQ users
- Send an email to us and let us know how happy you are
- Introduce other users to Linux and help them when they have problems.

Chapter 8. Miscellaneous

8.1. Useful Resources

- IP Masquerade Resource page Will have all the current information for setting up IP Masquerade on 2.0.x, 2.2.x, and even old 1.2 kernels!
- Juan Jose Ciarlante's WWW site (mirror) who is one of the current Linux IP Masquerade maintainers.

- IP Masquerade mailing list Archives contains the recent messages sent to the mailing lists.
- David Ranch's Linux page including the TrinityOS Linux document and current versions of the IP-MASQ-HOWTO.. Topics such as IP MASQ, strong IPFWADM/IPCHAINS rulesets, PPP, Diald, Cablemodems, DNS, Sendmail, Samba, NFS, Security, etc. are covered.
- The IP Masquerading Applications page: A comprehensive list of applications that work or can be tuned to work through a Linux IP masquerading server.
- For users setting up IP Masq on MkLinux, email Taro Fukunaga at tarozax@earthlink.net for a copy of his short MkLinux version of this HOWTO.
- IP masquerade FAQ has some general information
- Paul Russel's http://www.netfilter.org/ipchains/ doc and its possibly older backup at Linux IPCHAINS HOWTO. This HOWTO has lots of information for IPCHAINS usage, as well as source and binaries for the ipchains tool.
- X/OS Ipfwadm page contains sources, binaries, documentation, and other information about the `ipfwadm` package
- Check out the GreatCircle's Firewall mailing list for a great resource about strong firewall rulesets.
- The LDP Network Administrator's Guide is a MUST for the beginner Linux administrator trying to set up a network.
- The Linux NET HOWTO is also another comprehensive document on how to setup and configure Linux networking.
- Linux ISP Hookup HOWTO and Linux PPP HOWTO gives you information on how to connect your Linux host to the Internet
- Linux Ethernet-Howto is a good source of information about setting up a LAN running over Ethernet.
- Donald Becker's NIC drivers and Support Utils
- You may also be interested in Linux Firewalling and Proxy Server HOWTO
- Linux Kernel HOWTO will guide you through the kernel compilation process
- Other Linux HOWTOs such as Kernel HOWTO
- Posting to the USENET newsgroup: comp.os.linux.networking

8.2. Linux IP Masquerade Resource

The Linux IP Masquerade Resource is a website dedicated to Linux IP Masquerade information also maintained by Ambrose Au. It has the latest information related to IP Masquerade and may have information that is not being included in the HOWTO.

You may find the Linux IP Masquerade Resource at the following locations:

- http://ipmasq.webhop.net/, Primary Site, redirected to http://ipmasq.webhop.net/
- http://ipmasq2.webhop.net/, Secondary Site, redirected to http://www.e-infomax.com/ipmasq

---

8.3. Thanks to the following supporters..

In Alphabetical order:

- Gabriel Beitler, gabrielb@voicenet.com on providing section 3.3.8 (setting up Novell)
- Juan Jose Ciarlante, irriga@impsat1.com.ar for contributing his work on the IPMASQADM port forward tool, the 2.1.x and 2.2.x kernel code, and the original LooseUDP patch, etc.
- Steven Clarke, steven@monmouth.demon.co.uk for contributing his IPPORTFW IP port forwarder tool
- Andrew Deryabin, djsf@usa.net for contributing his ICQ MASQ module
- Ed Doolittle, dolittle@math.toronto.edu for suggestions to `-V` option in the`ipfwadm` command for improved security
- Matthew Driver, mdriver@cfmeu.asn.au for helping extensively on this HOWTO, and providing section 3.3.1 (setting up Windows 95)

- Ken Eves, ken@eves.com for the FAQ that provides invaluable information for this HOWTO
- John Hardin, jhardin@impsec.org for his PPTP and IPSEC forwarding tools
- Glenn Lamb, mumford@netcom.com for the LooseUDP patch
- Ed. Lott, edlott@neosoft.com for a long list of tested system and software
- Nigel Metheringham, Nigel.Metheringham@theplanet.net for contributing his version of the IP Packet Filtering and IP Masquerading HOWTO, which makes this HOWTO a better and in-depth technical document section 4.1, 4.2, and others
- Keith Owens, kaos@ocs.com.au for providing an excellent guide on ipfwadm section 4.2 on correction to `ipfwadm -deny` option which avoids a security hole, and clarified the status of `ping` over IP Masquerade
- Michael Owings, mikey@swampgas.com for providing the section about CU-SeeMe and Linux IP-Masquerade Teeny How-To
- Rob Pelkey, rpelkey@abacus.bates.edu for providing section 3.3.6 and 3.3.7 (setting up MacTCP and Open Transport)
- Harish Pillay, h.pillay@ieee.org for providing section 4.5 (dial-on-demand using Diald)
- Mark Purcell, purcell@rmcs.cranfield.ac.uk for providing section 4.6 (IPautofw)
- David Ranch, dranch@trinnet.net for maintaining the HOWTO, helping with the Linux IP Masquerade Resource Page, the TrinityOS document, ..., too many to list here :-)
- Paul Russell, rusty@linuxcare.com.au for all his work on IPTABLES, IPCHAINS, the IP Masquerade kernel patches in general, etc. The man is a IP NATing fool!
- Ueli Rutishauser, rutish@ibm.neton providing section 3.3.9 (setting up OS/2 Warp)
- Steve Grevemeyer, grevemes@tsmservices.com for taking over the IP Masq Applications page from Lee Nevo and updating it to a full DB backend.
- Fred Viles, fv@episupport.com for his patches on proper port forarding of FTP.
- John B. (Brent) Williams, forerunner@mercury.net on providing section 3.3.7 (setting up Open Transport)
- Enrique Pessoa Xavier, enrique@labma.ufrj.bron the BOOTp setup suggestion
- All the users on the IP-MASQ email list, masq@indyramp.com for their help and support for all the new Linux MASQ users.
- Other code and documentation developers of IP Masquerade for this great feature
- Delian Delchev, delian@wfpa.acad.bg
- David DeSimone (FuzzyFox), fox@dallas.net
- Jeanette Pauline Middelink, middelin@polyware.iaf.nl
- Miquel van Smoorenburg, miquels@q.cistron.nl
- Jos Vos, jos@xos.nl
- And others who I may have failed to mention here (please let me know)
- All users sending feedback and suggestions to the mailing list, especially those who reported errors in the document and the clients who are supported and not supported
- We apologize if we have omitted any important names, not included information that some fellow users have sent us yet, etc. There were many suggestions and ideas sent but there wasn't enough time to verify and integrate these changes. David Ranch is constantly trying his best to incorporate all of the information sent to me into the HOWTO. I thank you for the effort, and I hope you understand our situation.

8.4. Reference

- Original IP masquerade FAQ by Ken Eves
- IP masquerade mailing list archive by Indyramp Consulting
- IP Masquerade WWW site by Ambrose Au
- Ipfwadm page by X/OS
- Various networking related Linux HOWTOs
- Some topics covered in TrinityOS by David Ranch

8.5. ChangeLOG

TO do - HOWTO:

- Add the scripted IPMASQADM example to the Forwarders section. Also confirm the syntax.
- Add a little section on having multiple subnets behind a MASQ server
- Confirm the IPCHAINS ruleset and make sure it is consistant with the IPFWADM ruleset

TO DO - WWW page:

- Update the PPTP patch on the masq site
- Update the portfw FTP patch

Changes from 05/22/05 to 11/13/05

- 11/13/05 - Fix a bug where the PORTFW example rule in section 6.7 was incorrect. Updated the IPTABLES PORTFW section to include state tracking for the pre-routing rule, added a cross-reference to the PORTFW FAQ entry, and reduced some duplicate PORTFW examples in different chapters of the HOWTO. Thanks to Thomas Zajic for bringing this to my attention.
- 10/23/05 - Updated the dynamic IP FAQ section to give complete examples on how to re-run the rc.firewall-* scripts for various different DHCP clients
- 10/19/05 - Updated the HOWTO to be very clear on loading the various rc.firewall-* rulesets (there are 6 of them in this HOWTO both simple and stronger versions for IPTABLES, IPCHAINS, and IPFWADM) files vs. loading a generic rc. firewall file. I also updated the troubleshooting section to reflect this possibly confusing point.
- 05/27/05 - Updated the Multiple NAT situation to include ProxyARP solutions
- 05/26/05 - Clarified the section for IPMASQ on multiple internal LAN segments

Changes from 05/03/05 to 05/22/05

- 05/22/05 - Updated the rc.firewall-iptables-stronger ruleset to 0.87s. Removed the unused drop-and-logit chain as it was only later being deleted anyway. Thanks to Matthew Concannon for this one.
- 05/21/05 - Updated the Multiple-IPs FAQ entry a bit

Changes from 04/17/05 to 05/03/05

- 05/03/05 - Updated the rc.firewall-iptables-stronger ruleset to fix a typo

Changes from 03/19/04 to 04/17/05

- 04/30/05 - Updated the IP address for unc.metalab.org and published the HOWTO to the web.
- 12/18/04 - Added some comments in the IPTABLES, IPCHAINS, and IPFWADM rulesets why the default policy is ACCEPT and not something like DROP.
- 07/24/04: Renamed the rc.firewall-2.4/2.2/2.0-* rulesets to rc.firewall-iptables/ipchains/ipfwadm-*. This change better reflects that these rulesets can run on different kernel versions (such as 2.6.x). Updated the rc.firewall-iptables-stronger ruleset to 0.85s to fix an improper /24 netmask for the INTIP variable.
- 04/10/04: Updated the rc.firewall-2.4-stronger ruleset to use the 192.16.0.x network instead of 192.168.1.x network to better align with the rest of the HOWTO
- 04/04/04: Added that Redhat9 supports IPMASQ

Changes from 11/10/03 to 03/18/04

- 03/18/04: Added a sub-section for supporting multiple internal networks for IPTABLES
- 02/02/04: Updated some old jhardin rubyriver to impsec.org URLs
- 01/10/04: Updated the rc.firewall-2.4-stronger and 2.2 rulesets to make placement of PORTFW configs more obvious
- 01/01/04: Some systems require that the /etc/rc.d/init.d/firewall-2.* files be executable. Fixed. Thanks to Chris Carter and

others for the nudge.
- 01/01/04: Some systems require that the /etc/rc.d/init.d/firewall-2.* files be executable. Fixed. Thanks to Chris Carter and others for the nudge.
- 01/01/04: Added an additional chkconfig check on Redhat systems to make sure that the firewall will load upon init level change. Thanks to Chris Carter for the idea.
- 12/19/03: Updated the rc.firewall-2.4-stronger ruleset to 0.82. This new ruleset has a special ICMP filter to work around a Netfilter bug. Also, the drop-and-log-it chain has been renamed to reject-and-log-it since that's actually what it's doing. Thanks to Bart Martens for the recommendations.
- 12/13/03: Fixed some minor grammar issues. Thanks to Lawrence Berlinsk for pointing them out.
- 11/30/03: Updated the rc.firewall-2.4-stronger ruleset to 0.81s, the rc-firewall-2.2-stronger ruleset to 0.72s, and updated the rc.firewall-2.0-stronger ruleset to 0.72s (never had a version # before). These changes reflect either the ruleset not having strong enough comments or allowing all traffic destined to the MASQ server itself from being protected. It's recommend that if you want to enable access to servers running on the MASQ server itself (http, ssh, etc.), selectively enable them under the OPTIONAL INPUT section.
- 11/03/03: Updated the rc.firewall-2.2-stronger ruleset where an INTLAN rule that was allowing traffic from ANY IP address instead of the proper INTIP IP address only. This aligns the IPCHAINS ruleset with the IPTABLES and IPFWADM ruleset examples
- 11/10/03: Deleted all kernelnotes.org URLS (juanjox URLs)

Changes from 06/22/03 to 11/09/03

- 10/25/03: Fixed a dead RFC1918 URL in section 3.3. Thanks to Mark Sobell for the report.
- 07/07/03: Added the "reducing-masq-log" FAQ entry to help people reduce the size of their firewall logs.
- 06/27/03: Updated the rc.firewall-2.4-stronger ruleset to 0.80s. Added a DISABLED ip_nat_irc kernel module section, changed the default of the ip_conntrack_irc to NOT load by default, and added additional kernel module comments.
- 06/27/03: Updated the rc.firewall-2.4 ruleset to 0.75. Added additional iptables kernel module comments.
- 06/24/03: Added Debian 3.0 to the supported distro list
- 06/23/03: Change the PMTU URLs to point to Phil's primary www site

Changes from 05/26/03 to 06/22/03

- 06/22/03: Updated the various Indyramp MASQ email URLs again as things seemed to have changed. Again.
- 06/21/03: Rewrote the MTU FAQ section to be more clear, include specific information of the problems, and also fixed a bad typo for PPPoE users who were trying to configure "--clamp-mss-to-mtu" when it should have been "--clamp-mss-to-pmtu" (missing the p in pmtu).
- 06/13/03: Added kernel info for Mandrake 8.1
- 06/02/03: Fixed a typo where extended 2.2.x kernel checks for IPMASQ functionality was using "cat" and not "ls"

Changes from 04/08/03 to 05/26/03

- 05/26/03: updated the firewall rulesets: rc.firewall-2.4 (to 0.74), rc.firewall-2.2 (to 1.22), rc.firewall-2.4-stronger (to 0.79s), and rc.firewall-2.2-strongerw (to 0.71s) to use modprobe instead of insmod.
- 05/26/03: Added how to dump IPTABLES MASQ entries in the Accounting FAQ section
- 05/26/03: Added Clamp-MSS recommendations to the MTU faq section
- 05/26/03: Added additional troubleshooting steps in Section 5 when the MASQ client cannot ping the MASQ server.
- 05/26/03: Added additional traffic shaping / traffic limiter URLs to the SHAPING FAQ entry
- 05/26/03: Renamed the "IPROUTE2" FAQ entry to "Souce Routing"; Added IPTABLES examples to the section; fixed an incorrect IP address of 62123.123.123.123
- 05/25/03: Fixed a SGML script that was improperly converting ampersands for the downloadable firewall-* and rc.firewall-* scripts. Also caught a SGML ampersand bug in a comment section of the rc.firewall-2.0 file
- 05/25/03: Deleted several dead links: ftp.gts.cz, novell.com LWP5, Old Juanjox mirror (geocities), old ipmasq2.webhop.net URL, old zzdmacka NAT information URL, old linux.org/uk/VERSION url, old netfilter.samba.org URLs (no longer a netfilter mirror - redirect), old Activision BattleZone DLL url, old iproute2 (rpms, ras.ru, donlug, dontsk, tusur, waaug, etc.) urls, old rlynch ipautofw mirror

- 05/25/03: Updated several URLs: suse/proxy_suite/, www.indyramp.net URLs, several urls with " ~ " in it became ~732 for some reason, updated all of the jhardin URls to point from wolfnet.com to impsec.org, updated all LDP urls (linuxdoc.org to tldp.org), IPCHAINS patches for 2.0.x kernels, metalab to tldp.org, winfiles.com to download.com, Microsoft technet article 172227, Oidentd, mumford LooseUDP URL, 2.2.x PORT-FTP URL, IRQTUNE url, midentd URL
- 05/25/03: Pending updates from remote webmasters: Indyramp EQL URL, insecurity.net sidentd
- 05/25/03: Lots of little updates like:: updated the Intro section verbage a little to reflect BETA kernels and not OLD kernels; Updated the Forward section (not PORTFW) to be a little more generic; Added a link in the Forward to the IPMASQ email list; Updated the dates in the copyright notice;
- 05/24/03: Updated the "Current Status" to add the remark that some programs have been updated to use NAT-friendly protocols and thus special NAT modules are no longer required
- 05/24/03: Updated the 2.4 Requirements section: deleted a duplicate line (true 1:1 NAT); cleaned some addition things up; Added CuSeeme to the 2.4 ported list
- 05/24/03: Updated the 2.2 / 2.0 Requirements section: Deleted the reference to the obsoltele IPMASQ ICQ module; Updated the link for the LooseUDP URL;
- 05/24/03: Updated the Compiling Linux 2.2.x / 2.0.x section: Deleted the recommendations to load the rc.firewall ruleset via rc.local. This should come later in the HOWTO and offer other methods for different Linux distributions
- 05/24/03: Updated the ICQ Application section to say that these steps are /not/ required for modern ICQ clients. I've left this section in the HOWTO to demonstrate a large PORTFW example
- 05/24/03: Made some of the FAQ entries more kernel version generic and also deleted the 2.0.x "upgrades-cont.html" FAQ entry as it was basically a duplicate
- 05/24/03: Updated the LooseUDP game section to explain how it works, explain how much of this was properly solved under the stateful IPTABLES systtem, and also say that it is NOT available for the 2.4.x kernels. If IPTABLES's stateful UDP tracking doesn't work for, you're probably out of luck.
- 05/24/03: Mentioned in the FAQ section that MASQ timers are NOT adjustable under IPTABLES
- 05/24/03: Vastly expanded the packet firewall log FAQ entry and finally added a IPTABLES packet log description section. I also aligned the IPCHAINS example to match the IPFWADM entry
- 04/11/03: Fixed a incorrect echo statement saying the IPTABLES policy was being set to REJECT and not DROP.

Changes from 01/31/03 to 04/08/03

- 04/08/03: Added additional formatting and the "ip_masquerade" /proc entry into Section 3.2. This helps users determine if their kernel is MASQ-ready.
- 03/08/03: Added the EXTIP variable to the 2.4.x PORTFW example as several people were trying to use this with the BASIC ruleset and I had assumed they were using the STRONGER ruleset. Thanks to Greg Lukins for bringing this to my attention.
- 03/08/03: Added Distros to the MASQ compatibility list: Mandrake, Gentoo
- 02/08/03: Forgot to update the VERSION number for the rc.firewall-2.4-stronger rulese. Added some additional formatting
- 02/01/03: Added additional checking in the kernel compiling section to understand if your kernel supports IPMASQ via modules or by being statically compiled in.

Changes from 01/12/03 to 01/31/03

- 01/31/03: Doh. I should have read my own comments. I've reversed the 2.4.x. policy settings from REJECT back to DROP. REJECT, for some lame reason, is not a legal policy. The recommended REJECT action is still carried out via the "drop-and-log-it" user chain.
- 01/30/03: Updated the Multiple-IPs FAQ entry to better describe how users that want to put external IPs behind a Linux router. Added additional URLs and cleaned up the text a bit too.
- 01/30/03: Updated the 2.4.x requirement section to reflect more of the pros of IPTABLES as well as updated the update status of some old legacy 2.2.x modules
- 01/30/03: Added an additional FAQ entry that clearly explains what the ipchains.o module can and CANNOT do on 2.4.x. kernels
- 01/28/03: Extensively updated the 2.4.x kernel compilation section to reflect a 2.4.20 kernel with IPTABLES 1.2.7a. The section also reflects the new methods to compile IPTABLES, apply Patch-O-Matic patches, and also included lots of example output too.
- 01/28/03: Updated the kernel compiling section to be a little more clear on how different Linux distros can have different

kernels (modules vs. monolithic)
- 01/17/03: Fixed a major issue where the rc.firewall-2.2-stronger ruleset was referencing missing executable variables. This was taken from the 2.4-stronger ruleset but I guess I forgot to finish it off. Fixed. Thanks to Samuel Kim for catching this!
- 01/17/03: Fixed an issue where the rc.firewall-2.2-stronger's commented HTTP section was missing the "-p tcp" option. Thanks to Samuel Kim for catching this!
- 01/16/03: Updated the URL for DJSF's ICQ module
- 01/16/03: Changed the default policy and drop chain from DENY to REJECT on both IPTABLES rulesets and on the advanced IPFWADM rulset. Thanks to Jonathan Hutchins for bringing this to my attention.
- 01/16/03: Fixed a typo in the commented out HTTPd OUTPUT section of the rc.firewall-2.2-s ruleset
- 01/13/03: Updated the IPMASQ www site URL from ipmasq.cjb.net to ipmasq.webhop.net. CJB started to change their policies so we switched.
- 01/13/03: Added to the 2.4.x Requirements section that IPTABLES v1.2.7a is out and recommended.
- 01/13/03: Added an additional test item to the "Test Section - Section 5" for versions of IPTABLES that are too old. I also cleaned up this section to read easier.
- 01/13/03: Updated the rc.firewall-2.4-stronger ruleset to include commented rules to allow in HTTP traffic to a local HTTP server. Also added a rule comment in the FORWARD section to help users know where to put PORTFW commands.
- 01/13/03: Updated the rc.firewall-2.2-stronger ruleset to include commented rules to allow in HTTP traffic to a local HTTP server. Also added a rule comment in the FORWARD section to help users know where to put PORTFW commands.
- 01/13/03: Clarified the PORTFW section to help users better understand where the PORTFW commands should go in the rc. firewall rulesets. I also cleaned up this section to read a little better.

Changes from 12/13/02 to 01/12/03

- 01/03/03: Added Redhat 7.3 and 8.0 to the compatibility chart.
- 01/03/03: Fixed various typos. Thanks to Gabriel Withington for the sharp eye.
- 12/22/02: Updated the 2.2.x H.323 kernel patch URL. Thanks to Maxime Plante for pointing this out.
- 12/22/02: Updated the 2.4.x kernel compiling section to let users know that most modern kernels don't need IPTABLES Patch-o-matic patches to be applied except to fix bugs or add additional functionality.

Changes from 10/20/02 to 12/13/02

- 11/27/02: Fixed the init.d scripts to point the header to the correct config file. This must be due to newer versions of "chkconfig" doing better checking. Please note that this might still be a problem for the rc.firewall-2.?-stronger rulesets. Thanks to Joris Van Puyenbroeck for the heads up.
- 11/25/02: Updated all the firewall comments to reflect that PPPoE users need to user the "ppp0" logical interface as their external interface instead of the physical interface such as "eth0". Thanks to Meng Cheah for the nudge.
- 11/13/02: Updated the URL for the Donald Becker based NIC drivers. Thanks to Bruce Gorgon for the heads up.
- 11/01/02: Added a new FAQ section that covers redirection of local INTERNAL traffic to internal PORTFWed servers
- 11/01/02: Updated the PORTFW section to be a little more clear.

Changes from 04/19/02 to 10/20/02

- 09/29/02: Fixed a stray incorrect IP address pointing to metalab.unc.edu
- 08/29/02: Fixed a typo in the firewall-2.2 startup script which was starting the 2.4 firewall and not the 2.2. version. Thanks to Jean-Marc Vanel for catching this.
- 08/25/02: Updated the rc.firewall-2.2-stronger and rc.firewall-2.2 scripts to use shell environment variables.
- 07/09/02: Updated the FTP PORTFW section to be more readible
- 07/06/02: Replaced all the filewatcher.org URLs with netfilter.org URLs
- 06/12/02: Changed some of the formatting to try and help newbies better understand that the "\" character is used as a continuation of the previous line.
- 06/12/02: Updated the IP address of metalab.unc.edu in Section 5. Thanks to Pete Trachy for bringing this to my attention but please note that even major sites like Metalab change their IPs, subnets, or even ISPs from time to time.
- 06/02/02: Updated the rc.firewall-2.4 ruleset to include a commented option for NATing IRC DCCs, added the use of more environment vars, and added additional formatting.

- 05/18/02: Added some extra # lines the commented section of the the rc.firewall-2.4-stronger ruleset to better serve Cut and Paste users.
- 05/04/02: - Updated the various PPTP MASQ links to point to a valid URL. Also updated the HOWTO to reflect that PPTP is now supported on the 2.4.x kernels.
- 05/03/02: - Updated the 2.4.x kernel requirements section to point out that IPCHAINS compatibility under 2.4.x kernels isn't very good. If you want to use IPMASQ under a 2.4.x kernel, you should use IPTABLES rules only.

Changes from 01/05/02 to 04/19/02 - v2.00.041902 pubsished to the LDP

- 04/01/02: - Updated the rc.firewall-2.4-stronger ruleset to denote and disable internal DHCP server support on the OUTPUT rules
- 02/09/02: - Added Redhat-style init.d scripts to start the rc.firewall files
- 02/09/02: - Updated all the various chapters to use human readable file names vs. things like x2623.html.
- 02/09/02: - Expanded the IPMASQ accounting section
- 02/04/02: - Deleted an extra "$" from the PORTFW variable in section 6.7.1
- 01/31/02: - Updated the URLs for the PPPd and Diald homepages
- 01/26/02: - Fixed some typos and added a LooseUDP clarification to tell users to read the example rc.firewall-2.2 ruleset comments on how to enable LooseUDP.
- 01/08/02: - Made some slight clarifications to IP Alias support

Changes from 11/19/01 to 01/05/02 - 010502 pubsished to the LDP

- 01/05/02: - Added disabled rules to the rc.firewall-2.4-stronger ruleset to support INTERNAL DHCP server and EXTERNAL access to a WWW server running on the MASQ machine.
- 01/05/02: - Added required changes to the loading of the ip_conntrack_ftp module if people PORTFW to non-standard FTP ports.
- 01/05/02: - Added an example in the 2.4.x PORTFW section on how to REDIRECT internal traffic back to an INTERNAL server. This is the same as running REDIR under 2.2.x and 2.0.x kernels.
- 01/05/02: - Added Juanjox mirror URLs to the HOWTO.
- 01/04/02: - Clarified and cleaned up the ICQ PORTFW section; Added thoughts on the ip_masq_icq, PORTFW, and SOCKS solutions
- 01/05/02: - Added Slackware 8.0 to the supported list.
- 01/04/02: - Fixed some spelling mistakes in the 2.4 and 2.2 rulesets. Thanks to Michael Ott for the sharp eye.
- 12/19/01: - Fixed a minor comment typo in the rc.firewall-2.4 file. Thanks to Bruno Negrao for this one.
- 12/02/01: - Fixed some minor version typos in the 2.4.x rc.firewall ruleset; Added a missing $PORTFWIF variable for the 2.4.x PORTFW example. Thanks to Neil Bunn for the errata.
- 11/25/01: - Expanded on the ipchains module conflict error messages in Section 5
- 11/23/01: - Updated the HOWTO to reflect a new PPTP kernel module for the 2.4.x kernels
- 11/19/01: - Clarified the PPTP supports for 2.4.x kernels

Changes from 08/26/01 to 11/18/01 - 111801 published to the LDP

- 11/12/01: - updated various comments to reflect new versions:linux 2.4.14, iptables 1.2.4, and linux 2.2.20.
- 11/12/01: - Added the rc.firewall-2.4-stronger ruleset to the HOWTO, updated the 2.4.x kernel and IPTABLES compiling steps to reflect 2.4.14 and 1.2.4.
- 11/10/01: - Added the directly downloadable versions of the 2.4, 2.4-stronger, 2.2, 2.2-stronger, 2.0, and and 2.0.x-stronger rulesets to the WWW.
- 11/10/01: - Updated the 2.4.x PORTW example to add the missing FORWARD option.
- 11/10/01: - Updated the DSL-HOWTO link in the HOWTO
- 10/27/01: - Updated the network diagram in section 2.5 to be a little more verbose.
- 09/18/01: - Fixed some broken reference links pointing to the respective 2.4.x, 2.2.x, and 2.0.x kernel compiling recommendations.
- 09/16/01: - Cleaned up and updated the PORTFW section to also include PREROUTING examples for 2.4.x kernels.
- 09/13/01: - Updated the IPTABLES simple rc.firewall ruleset to 0.62. This fixed a typo on the MASQ enable line that used

eth0 instead of $EXTIF. Thanks to Hafi for reporting this.
- 09/07/01: - It seems that most people who are getting IPCHAINS and IPTABLES conflicts are running Redhat 7.1. I have updated section 5 on how to fix this. Thanks to Jason Wenzel for helping me with this.
- 09/07/01: - Noted that IPTABLES v1.2.3 is current version. All versions less than v1.2.3 have an FTP module bug that can bypass strong firewall rulesets. Please upgrade your copy of IPTABLES now.
- 09/07/01: - Created version numbers for the simple rc.firewall rulesets (IPTABLES - v0.61) (IPCHAINS - v1.01) (IPFWADM - v2.01). and cleaned up some of the comments in each section.
- 09/07/01: - Added rules to the simple rc.firewall rulesets to flush the various tables. In addition to this, I have added the use of environment variables and more echo statements in the rulesets to make things easier to edit and monitor. Thanks to Ian Bishop for the good idea.
- 09/07/01: - Added the use of EXTIF and INTIF interface variables in each of the rc.firewall and partial firewall rulesets for better clarity (similar to how TrinityOS has been doing for a while now). Thanks to Sean McKeon for the nudge.
- 09/07/01: - Fixed a typo in the UNIX client configuration section where the network broadcast was 192.168.0.25 instead of .255.

Changes from 2.01 to 2.05 - 08/26/01

- 08/19/01: - Added an additional testing step in Section5 to make sure the rc.firewall file loads ok. Thanks to Steven Levis for the good idea.
- 08/15/01: - Change the reference for the /etc/hosts file from RFC952 to RFC1035. Thanks to Michael F. Maggard for the correction.

Changes from 1.96 to 2.01 - 08/12/01

- 08/12/01: - Updated the basic IPTABLES ruleset to 0.60 which fixed a major issue where all MASQed packets were being dropped. Ultimately, I forgot to add a rule to ACCEPT correct packets through the forwarding chain.
- - Added an additional rule to log all bogus FORWARD packets
- - Load the FTP nat modules now by default
- - Changed the load order of some of the kernel modules to not create bogus error messages
- - Added an IPTABLES section on how to MASQ specific hosts vs. an entire subnet
- - Added more MASQ-client compatible operating systems

- 07/19/01: - The advanced IPCHAINS example for forwarding between multiple interfaces was missing the critital "-j ACCEPT" to actually let the packets flow. Thanks to Shingo Yamaguchi for catching this.

Changes from 1.96 to 2.00 - 06/10/01

- 06/21/01: Updated Section 5 (Testing Section) to add an additional test to help users troubleshoot their MASQ setup. There are now a total of -11- tests. 06/16/01: Updated the intro History section at the beginning of the HOWTO. 06/14/01: Added mirror Netfilter and IPCHAINs mirror URLs 06/13/01: Updated the H.323 URL
- 06/10/01: Double DOH! The simple rc.firewall script for the 2.4 kernels had two major errors in it. The new version is far more informative and even works! I am continuing to go through the HOWTO and cleaning things up but I'm not done quite yet.
- 06/02/01: Updated the lists of known compatible MASQ'ed operating systems (Windows M3, Linux 2.3, 2.4, etc) Made more references to DHCP and DNS in the various different MASQ client configuration guides.
- 04/12/01: Thanks to the Joshua X and the other people at Command Prompt, Inc. for the port of the HOWTO from LinuxDoc to DocBook. Add email list URL to line 126

Changes from 1.90 to 1.95 - 11/11/00

- A BIG thanks to the Joshua X and the other people at Command Prompt, Inc. for the port of the HOWTO from LinuxDoc to DocBook.
- Added a quick upfront notice in the intro that running a SINGLE NIC in MASQ mutliple ethernet segments is NOT recommended and linked to the relivant FAQ entry. Thanks to Daniel Chudnov for helping the HOWTO be more clear.

- Added a pointer in the Intro section to the FAQ section for users looking for how MASQ is different from NAT and Proxy services.
- Reordered the Kernel requirements sections to be 2.2.x, 2.4.x, 2.0.x
- Expanded the kernel testing in Section 3 to see if a given kernel already supports MASQ or not.
- Reversed the order of the displayed simple MASQ ruleset examples (2.2.x and 2.0.x)
- Cleaned up some formatting issues in the 2.0.x and 2.2.x rc.firewall files
- Noted in the 2.2.x rc.firewall that the defrag option is gone in some distro's proc (Debian, TurboLinux, etc)
- Added a NOTE #3 to the rc.firewall scripts to include instructions for Pump. Thanks to Ross Johnson for this one.
- Cleaned up the simple MASQ ruleset examples for both the 2.2.x and 2.2.x kernels
- Updated the simple and stronger IPCHAINS and IPFWADM rulesets to include the external interface names (IPCHAINS is -i; IPFWADM is -W) to avoid some internal traffic MASQing issues.
- Vastly expanded the Section 5 (testing) with even more testing steps with added complete examples of what the output of the testing commands should look like.
- Moved the H.323 application documentation from NOT supported to Supported. :-)
- Reordered the Multiple LAN section examples (2.2.x then 2.0.x)
- Made some additional clarifications to the Multiple LAN examples
- Fixed a critical typo with multiple NIC MASQing where the network examples had the specified networks reversed. Thanks to Matt Goheen for catching this.
- Added a little intro to MFW in the PORTFW section.
- Reveresed the 2.0.x and 2.2.x sections for PORTFW
- Updated the news regarding PORTFWing FTP traffic for 2.2.x kernels

```
  NOTE:  At this time, there *IS* a BETA level IP_MASQ_FTP module
         for PORT Forwarding FTP connections 2.2.x kernels which also
supports
         adding additional PORTFW FTP ports on the fly without the
requirement
         of unloading and reloading the IP_MASQ_FTP module and thus breaking
         any existing FTP transfers.
```

- Added a top level note about PORTFWed FTP support
- Added a noted to the 2.0.x PORTFW'ed FTP example why users DON'T need to PORTFW port 20.
- Updated the PORTFW section to also mention that users can use FTP proxy applications like the one from SuSe to support PORTFWed FTP-like functionality. Thanks to Stephen Graham for this one.
- Updated the example for how to enable PORTFWed FTP to also include required configurations on how the ip_masq_ftp module is loaded for users who use multiple PORTs to contact multiple internal FTP servers. Thanks to Bob Britton for reminding me about this one.
- Added a FAQ entry for users who have embedded ^Ms in their rc.firewall file
- Expanded the FAQ entry talking about how MASQ is different from NAT and Proxy to include some informative URLs.
- Updated the explanation of the MASQ MTU issue and described the two main explanations for the issue.
- Clarified that the RFC, PPPoE should only require an MTU of 1492 though some ISPs require a setting of 1460. Because of this, I have updated the example to show an MTU of 1492.
- Broke out the Windows 9x sections into Win95 and Win98 as they use different settings (DWORD vs. STRING). I also updated the sections to be clearer and the Registry backup methods have been updated.
- Fixed a typo where the NT 4.0 Registry entries were backwards (Tcpip/Parameters vs. Parameters/Tcpip).
- Fixed an issue where the WinNT entry should have been a DWORD and not a STRING.
- A serious thanks goes out to Geoff Mottram for his various PPPoE and various Windows Registry entry fixes.
- Added an explict URL for Oident in the IRC FAQ entry
- Updated the FAQ section regarding some broken "netstat" versions
- Added new FAQ sections for MASQ accounting ideas and traffic shaping
- Expanded the IPROUTE2 FAQ entry on what Policy-routing is.
- Moved the IPROUTE2 URLs to the 2.2.x Kernel requirements section and also added a few more URLs as well.

- Corrected the "intnet" varible in the stronger IPCHAINS ruleset to reflect the 192.168.0.0 network to be consistent with the rest of the example. Thanks to Ross Johnson for this one.
- Added a new FAQ section for users asking about forwarding problems between multiple internal MASQed LANs.
- Added a new FAQ section about users wanting to PORTFW all ports from multiple external IP addresses to internal ones. I also touched on users who were trying to PORTFW all ports on multiple IP ALIASed interfaces and also noted the Bridge +Firewall HOWTO for DSL and Cablemodem users who have multiple IPs in a non-routed environment.
- Added Mandrake 7.1, Mandrake 7.2, and Slackware 7.1 to the supported list
- Added Redhat 7.0 to the MASQ supported distros. Thanks to Eugene Goldstein for this one.
- Fixed a mathematical error in the "Maximum Throughput" calculation in the FAQ section. Thanks to Joe White @ ip255@msn.com for this one.
- Fixed the Windows9x MTU changes to be a STRING change and not a DWORD change to the registry. Thanks to jmoore@sober.com for this one.
- Updated the comments in the 2.0.x rc.firewall script to note that the ip_defrag option is for both 2.0 and 2.2 kernels. Thanks to pumilia@est.it for this clarification.

Changes from 1.85 to 1.90 - 07/03/00

- Updated the URL for TrinityOS to reflect its newest layout
- Caught a typo in the IPCHAINS rulesets where I was setting "ip_ip_always_defrag" instead of "ip_always_defrag"
- The URL to Taro Fukunaga was invaild since it was using "mail:" instead of "mailto:"
- Added some clarification to the "Masqing multiple internal interfaces" where some users didn't understand why eth0 was referenced multiple times.
- Fixed another "space after the EXTIP variable" bug in the stronger IPCHAINS section. I guess I missed one.
- In Test #7 of Section 5, I referred users to go back to step #4. That should have been step #6.
- Updated the kernel versions that came with SuSe 5.2 and 6.0
- Fixed a typo (or vs. of) in Section 7.2
- Added Item #9 to the Testing MASQ section to refer users who are still haing MASQ problems to read the MTU entry in the FAQ
- Improved the itemization in Section 5
- Updated the IPCHAINS syntax to show the MASQ/FORWARD table. Before, it was valid to run "ipchains -F -L" but now only "ipchains -M -L" works.
- Updated the LooseUDP documentation to reflect the new LooseUDP behavior in 2.2.16+ kernels. Before, it was always enabled, now, it defaults to OFF due to a possible MASQed UDP port scanning vulnerability. I updated the BASIC and SEMI-STRONG IPCHAINS rulesets to reflect this option.
- Updated the recommended 2.2.x kernel to be 2.2.16+ since there is a TCP root exploit vulnerability on all lesser versions.
- Added Redhat 6.2 to the MASQ supported list
- Updated the link for Sonny Parlin's FWCONFIG to point to fBuilder.
- Updated the various examples of IP addresses from 111.222.333.444 to be 111.222.121.212 and within a valid IP address range
- Updated the URL for the BETA H.323 MASQ module
- Finally updated the MTU FAQ section to help out PPPoE DSL and Cablemodem users. Basically, Section 7.15 now reflects the fact that users can also change the MTU settings of all of their INTERNAL machines to solve the dreaded MASQ MTU issue.
- Added a clarification to the PORTFW section that PORTFWed connections which work for EXTERNAL clients but will not work for INTERNAL clients. If you also need INTERNAL portfw, you will need to also implement the REDIR tool as well. I also noted that this issue is fixed in the 2.4.x kernels with Netfilter.
- I also added a technical explanation from Juanjo to the end of the PORTFW section to why this senario doesn't work properly.
- Updated all of the IPCHAINS URLs to point to Paul Rusty's new site at http://www.netfilter.org/ipchains/
- Updated Paul Rustys email address
- Added a new FAQ section for users whose connections remain idle for a long period of time and PORTFWed connections no longer work.
- Updated all the URLs to the LDP that pointed to metalab.unc.edu to the new site of linuxdoc.org
- Updated the Netfilter URLs to point to renamed HOWTOs, etc.
- I also updated the status of the 2.4.x support to note that I *will* add full Netfilter support to this HOWTO and if the time

comes, then split that support off into a different HOWTO.
- Updated the 2.4.x Requirements section to reflect how NetFilter has changed compared to IPFWADM and IPCHAINS and gave a PROs/CONs list of new features and changes to old behaviors.
- Added a TCP/IP math example to the "My MASQ connection is slow" FAQ entry to better explain what a user should expect performance wise.
- Updated the HOWTO to reflect that newer versions of the "pump" DHCP client now can run scripts upon bringup, lease renew, etc.
- Updated the PORTFWing of FTP to reflect that several users say they can successfully forward FTP traffic to internal machines without the need of a special ip_masq_ftp module. I have made the HOWTO reflect that users should try it without the modified module first and then move to the patch if required.

Changes from 1.82 to 1.85 - 05/29/00

- Ambrose Au's name has been taken off the title page as David Ranch has been the primary maintainer for the HOWTO for over a year. Ambrose will still be involved with the WWW site though.
- Deleted a stray SPACE in section 6.4
- Re-ordered the compatible MASQ'ed OS section and added instructions for setting up a AS/400 system running on OS/400. Thanks to jaco@libero.it for the notes.
- Added an additional PORFW-FTP patch URL for FTP access if HTTP access fails.
- Updated the kernel versions for Redhat 5.1 & 6.1 in the FAQ
- Added FloppyFW to the list of MASQ-enabled Linux distros
- Fixed an issue in the Stronger IPFWADM rule set where there were spaces between "ppp_ip" and the "=".
- In the kernel compiling section for 2.2.x kernels, I removed the reference to enable "CONFIG_IP_ALWAYS_DEFRAG". This option was removed from the compiling section and enabled by default with MASQ enabled in 2.2.12.
- Because of the above change in the kernel behavior, I added the enabling of ip_always_defrag to all the rc.firewall examples.
- Updated the status of support for H.323. There are now ALPHA versions of modules to support H.323 on both 2.0.x and 2.2.x kernels.
- Added Debian v2.2 to the supported MASQ distributions list
- Fixed a long standing issue where the section that covered explict filtering of IP addresses for IPCHAINS had old IPFWADM syntax. I've also cleaned this section up a little and made it understandable.
- Doh! Added Juan Ciarlante's URL to the important MASQ resources section. Man.. you guys need to make me more honest than this!!
- Updated the HOWTO to reflect kernels 2.0.38 and 2.2.15
- Reversed the order shown to compile kernels to show 2.2.x kernels first as 2.0.x is getting pretty old.
- Updated the 2.2.x kernel compiling section to reflect the changed options for the latter 2.2.x kernels.
- Added a a possible solution for users that fail to get past MASQ test #5.

Changes from 1.81 to 1.82 - 01/22/00

- Added a missing subsection for /proc/sys/net/ipv4/ip_dynaddr in the stronger IPCHAINS ruleset. Section 6.5
- Changed the IP Masq support for Debian 2.1 to YES
- Reorganized and updated the "Masq is slow" FAQ section to include fixing Ethernet speed and duplex issues.
- Added a link to Donald Becker's MII utilities for Ethernet NIC cards
- Added a missing ")" for the 2.2.x section (previously fixed it only for the 2.0.x version) to the ICQ portfw script and changed the evaluation from -lt to -le
- Added Caldera eServer v2.3 to the MASQ supported list
- Added Mandrake 6.0, 6.1, 7.0 to the MASQ supported list
- Added Slackware v7.0 to the MASQ supported list
- Added Redhat 6.1 to the MASQ supported list
- Added TurboLinux 4.0 Lite to the MASQ supported list
- Added SuSe 6.3 to the MASQ supported list
- Updated the recommended stable 2.2.x kernel to be anything newer than 2.2.11
- In section 3.3, the HOWTO forgot how to tell the user how to load the /etc/rc.d/rc.firewall upon each reboot. This has now been covered for Redhat (and Redhat-based distros) and Slackware.

- Added clarification in the Windows WFWG v3.x and NT setup sections why users should NOT configure the DHCP, WINS, and Forwarding options.
- Added a FAQ section on how to fix FTP problems with MASQed machines.
- Fixed a typo in the Stronger firewall rulesets. The "extip" variabl cannot have the SPACE between the variable name and the "=" sign. Thanks to johnh@mdscomp.com for the sharp eye.
- Updated the compatibly section: Mandrake 7.0 is based on 2.2.14 and TurboLinux v6.0 runs 2.2.12

Changes from 1.80 to 1.81 - 01/09/00

- Updated the ICQ section to reflect that the new ICQ Masq module supports file transfer and real-time chat. The 2.0.x module still has those limitations.
- Updated Steven E. Grevemeyer's email address. He is the maintainer of the IP Masq Applications page.
- Fixed a few lines that were missing the work AREN'T for the "setsockopt" errors.
- Updated a error the strong IPCHAINS ruleset where it was using the variable name "ppp_ip" instead of "extip".
- Fixed a "." vs a "?" typo in section 3.3.1 in the DHCP comment section.
- Added a missing ")" to the ICQ portfw script and changed the evaluation from -lt to -le
- Updated the Quake Module syntax to NOT use the "ports=" verbage

Changes from 1.79 to 1.80 - 12/26/99

- Fixed a space typo when setting the "ppp_ip" address.
- Fixed a typo in the simple IPCHAINS ruleset. "deny" to "DENY"
- Updated the URLs for Bjorn's "modutils" for Linux
- Added verbage about NetFilter and IPTables and gave URLs until it is added to this HOWTO or a different HOWTO.
- Updated the simple /etc/rc.d/rc.firewall examples to notify users about the old Quake module bug.
- Updated the STRONG IPFWADM /etc/rc.d/rc.firewall to clarify users about dynamic IP addresses (PPP & DHCP), newer DHCPCD syntax, and the old Quake module bug.
- Updated the STRONG IPCHAINS /etc/rc.d/rc.firewall to ADD a missing section on dynamic IP addresses (PPP & DHCP) and the old Quake module bug.
- Added a note in the "Applications that DO NOT work" section that there IS a beta module for Microsoft NetMeeting (H.323 based) v2.x on 2.0.x kernels. There is NO versions available for Netmeeting 3.x and/or 2.2.x kernels as of yet.

Changes from 1.78 to 1.79 - 10/21/99

- Updated the HOWTO name to reflect that it isn't a MINI anymore!

Changes from 1.77 to 1.78 - 8/24/99

- Fixed a typo in "Section 6.6 - Multiple Internal Networks" where the -a policy was ommited.
- Deleted the 2.2.x kernel configure option "Drop source routed frames" since it is now enabled by default and the kernel compile option was removed.
- Updated the 2.2.x and all other IPCHAINS sections to notify users of the IPCHAINS fragmentation bug.
- Updated all of the URLs pointing at Lee Nevo's old IP Masq Applications page to Seg's new page.

Changes from 1.76 to 1.77 - 7/26/99

- Fixed a typo in the Port fowarding section that used "ipmasqadm ipportfw -C" instead of "ipmasqadm portfw -f"

Changes from 1.75 to 1.76 - 7/19/99

- Updated the "ipfwadm: setsockopt failed: Protocol not available" message in the FAQ to be clearer instead of making the user hunt for the answer in the Forwarders section.
- Fixed incorrect syntax in section 6.7 for IPMASQADM and "portfw"

Changes from 1.72 to 1.75 - 6/19/99

- Fixed the quake module port setup order for the weak IPFWADM & IPCHAINS ruleset and the strong IPFWADM ruleset as well.
- Added a user report about port forwarding ICQ 4000 directly in and using ICQ's default settings WITHOUT enabling the "Non-Sock" proxy setup.
- Updated the URLs for the IPMASQADM tool
- Added references to Taro Fukunaga, tarozax@earthlink.net for his MkLinux port of the HOWTO
- Updated the blurb about Sonny Parlin's FWCONFIG tool to note new IPCHAINS support
- Noted that Fred Vile's patch for portfw'ed FTP access is ONLY available for the 2.0.x kernels
- Updated the 2.2.x kernel step with a few clarifications on the Experiemental tag
- Added Glen Lamb's name to the credits for the LooseUDP patch
- Added a clarification on installing the LooseUDP patch that it should use "cat" for non-compressed patches.
- Fixed a typo in the IPAUTO FAQ section
- I had the DHCP client port numbers reversed for the IPFWADM and IPCHAINS rulesets. The order I had was if your Linux server was a DHCP SERVER.
- Added explict /sbin path to all weak and strong ruleset examples.
- Made some clarifications in the strong IPFWADM section regarding Dynamic IP addresses for PPP and DHCP users. I also noted that the strong rulesets should be re-run when PPP comes up or when a DHCP lease is renewed.
- Added references in the 2.2.x requirements, updated the ICQ FAQ section, and added Andrew Deryabin to the credits section for his ICQ MASQ module.
- Added some clarifcations to the FAQ section explaining why the 2.1.x and 2.2.x kernels went to IPCHAINS.
- Added a little FAQ section on Microsoft File/Print/Domain services (Samba) through a MASQ server. I also added an URL to a Microsoft Knowledge based document for more details.
- Added clarifications to the FAQ section that NO Debian distribution supports IP masq out of the box.
- Updated the supported MASQ distributions in the FAQ section.
- Added to the Aliased NIC section of the FAQ that you CANNOT masq out of an aliased interface.
- Wow.. never caught this before but the "ppp-ip" variable in the strong ruleset section is an invalid variable name! It has been renamed to "ppp_ip"
- In both the IPFWADM and IPCHAINS simple ruleset setup areas, I had a commented out section on enabling DHCP traffic. Problem is, it was below the final reject line! Doh! I moved both up a section.
- In the simple IPCHAINS setup, the #d out line for DHCP users, I was using the IPFWADM "-W" command instead of IPCHAINS's "-i" parameter.
- Added a little blurb to the Forwarders section the resolution to the famous "ipfwadm: setsockopt failed: Protocol not available" error. This also includes a little /proc test to let users confirm if IPPORTFW is enabled in the kernel. I also added this error to a FAQ section for simple searching.
- Added a Strong IPCHAINS ruleset to the HOWTO
- Added a FAQ section explaining the "kernel: ip_masq_new(proto=UDP): no free ports." error.
- Added an example of scripting IPMASQADM PORTFW rules
- Updated a few of the Linux Documentation Project (LDP) URLs
- Added Quake III support in the module loading sections of all the rc.firewall rulesets.
- Fixed the IPMASQADM forwards for ICQ
- 1.72 - 4/14/99 - Dranch: Added a large list of Windows NAT/Proxy alternatives with rough pricing and URLs to the FAQ.
- 1.71 - 4/13/99 - Dranch: Added IPCHAINS setups for multiple internal MASQed networks. Changed the ICQ setup to use ICQ's default 60 second timeout and changed IPFWADM/IPCHAINS timeout to 160 seconds. Updated the MASQ and MASQ-DEV email list and archive subscription instructions.
- 1.70 - 3/30/99 - Dranch: Added two new FAQ sections that cover SMTP/POP-3 timeout problems and how to masquerade multiple internal networks out onto different external IP addresses with IPROUTE2.
- 1.65 - 3/29/99 - Dranch: Typo fixes, clarifications of required 2.2.x kernel options, added dynamic PPP IP address support to the strong firewall section, additional quake II module ports, noted that the LooseUDP patch is built into later 2.2.x kernels and its from Glenn Lamb and not Dan Kegel, added more game info in the compatibility section.
- 1.62 - Dranch: Make the final first-draft changes to the doc and now announce it in the MASQ email list.
- 1.61 - Dranch: Made editorial changes, cleaned things up and fixed some errors in the Windows95 and NT setups.
- 1.58 - Dranch: Addition of the port forwarding sections; LooseUDP setup; Ident servers for IRC users, how to read firewall

logs, deleted the CuSeeme Mini-HOWTO since it is rarely used.

- 1.55 - Dranch: Complete overhaul, feature and FAQ addition, and editing sweep of the v1.50 HOWTO. Completed the 2.2.x kernel and IPCHAINS configurations. Did a conversion from IPAUTOFW to IPPORTFW for the examples that applied. Added many URLs to various other documentation and utility sites. There are so many changes.. I hope everyone likes it. Final publishing of this new rev of the HOWTO to the LDP project won't happen until the doc is looked over and approved by the IP MASQ email list (then v2.00).
- 1.50 - Ambrose: A serious update to the HOWTO and the initial addition of the 2.2.0 and IPCHAINS configurations.
- 1.20 - Ambrose: One of the more recent HOWTO versions that solely dealt with < 2.0.x kernels and IPFWADM.