Firewalls

An Introduction to Concepts and Implementation using Linux ***netfilter***

Andreas Rottmann

E-mail: e9926584@student.tuwien.ac.at

Abstract

This article first introduces the reader into the fundamental concepts of firewalls, illustrating how they help to increase system security by preventing certain types of attacks. Subsequently, an example firewall setup is described and implemented using the Linux ***netfilter*** *framework.*

Keywords

Linux, firewall, packet filter, netfilter, iptables

Contents

1 Introduction

1.1 Motivation

Today, it is considered standard for a computer to be connected to the Internet. As convenient and valuable the resources offered by Internet connectivity may be, it also means exposure of the machine to various risks, such as attacks by crackers, viruses, worms, trojans and other malign entities floating round the Net. A firewall can offer protection to a range of possible attacks, thus increasing the security of one or more machines connected to the Internet. A firewall, however, is not a measure to reach ultimate security, only part of the pool of measures that a system administrators can employ to secure the machines and networks they administer.

Stated simply, a firewall is a host whose main purpose is to protect a network by restricting access from "outside" to the network and often also restricting the protected network in its access of the "outside" (typically the Internet). A firewall can be used to[Wack et all. 1995]:

- *Increase network security* by limiting access to (potentially) vulnerable services. For example, an FTP server (which accepts plain-text passwords) might be OK to use from an internal WLAN, which has data-link level encryption, but is inherently insecure when used from the Internet or any other network that does not ensure encryption.
- *Control access* to site systems and services. In an unsecured network more services and hosts open to access than necessary. A firewall can be used to enforce a policy where only the access required for operation is possible.
- *Concentrate security* at a single host. By centralizing security policies in a single place, administration is simplified.
- *Enhance privacy* by using a firewall in combination with NAT (Network Address Translation) to hide information about the network behind the firewall, e.g. the network topology.
- *Gather information* about network (mis-) usage. Since all data sent to and received from the Internet passes through the firewall, valuable statistical and other information, e.g. suspicious activity can me collected.

1.2 Firewall Policy

Before a firewall can be implemented, there must exist a set of rules (the security policy) that lies the groundwork for building up the firewall. This minimally includes a list of services to be offered, the circumstances under which those are to be accessed (which users, from which machines, ...). This policy then is enforced by the firewall.

Also, one must decide the basic firewall design policy; it most commonly is either

- *permit* any service unless explicitly denied, or
- *deny* any service unless explicitly permitted.

Of these two, the latter is generally recommended, since it can also prevent access to unanticipated new services. However, due to its greater strictness, it is also more difficult to implement without impacting users and has difficulties with certain protocols (e.g. FTP and X Window) [Wack et all. 1995].

1.3 Types of Firewalls

A firewall can operate on different layers of the network stack. Most common are packet-filters, which operate on the network and transport layer. Those are often combined with application level gateways (e.g. filtering proxies). Examples can be found in [Wack et all. 1995].

1.3.1 Packet Filters

A packet filter works by inspecting the network packets at the network and transport layer. In the TCP/IP stack, this means filtering based on the IP header fields (e.g. source and sender address) and on the transport-level UDP and TCP ports. They cannot filter on content, since they have no knowledge about the protocol that the application uses (e.g. it is not possible implement a email virus filter). A packet filter is mostly transparent to the user, i.e. there is no special application setup required to use them.

1.3.2 Application Level Gateways

Application level gateways (also known as proxies), in contrast to packet filters, have inherent knowledge about the protocol they are gatewaying. Due

to that knowledge, a much greater extent of filter actions can be implemented. A HTTP proxy, for instance, is able to filter based on the URLs accessed and the content transmitted. Proxies are normally not transparent to the user[1].

2 Countering Attacks

There are several types of attacks one can try to prevent by deployment of a firewall, but also several attacks where a firewall is only a minor (or no) hindrance for the attacker. This section outlines some attacks and how a firewall can be used to prevent them and give reasons for why a firewall is no gain for certain kinds of attacks.

2.1 Viruses and Worms

The difference between a virus and a worm is that the latter acts as independent entity, replicating itself without needing to attach to a program for replication [2].

For instance, the MSBlaster worm replicated by exploiting a bug in the DCOM RPC implementation of MS Windows by sending specially crafted packages to the RPC service port (TCP Port 135 in most cases) [3]. This type of attack can easily be prevented by a packet filter firewall that blocks all incoming packages to (at least) the affected ports.

Viruses and Trojans, on the other hand, attach to a host program or pretend to serve a useful purpose and typically require user interaction to spread (or cooperation of insecure-by-default software such as MS Outlook (Express), which takes actions on receiving email without consulting the user first).

This means to avoid virus infection, one must either (a) make sure that both no insecure software is used *and* users act responsible in their handling of email, downloads, ..., or (b) prevent the virus from entering the network in the first place. Of course, (a) can be prohibitively difficult to achieve, since one cannot normally force users to act responsibly. So, best is to implement (b) and add (a) as a second barrier. To filter malware, a mere packet filter is not enough, as explained in Sect. 1.3.2. The two most probable ways for viruses and trojans to enter the network are email and download

[1] As shown in Sect. 4.2.1, Linux `netfilter` can be combined with Squid to form a transparent HTTP proxy.

[2] `http://en2.wikipedia.org/wiki/Computer_worm`

[3] `http://www.microsoft.com/technet/treeview/?url=/technet/security/bulletin/MS03-026.asp`

via HTTP/FTP. To block these ways (assuming one cannot simply disallow them), all the content must be piped through detection software integrated into an application level gateway for the respective protocol. See Sect. 4.2 for some solutions available under Linux.

2.2 Denial of Service Attacks

Denial of service (DoS) attacks can happen either at the network level or at application level.

At the network level the attacker sends specially crafted packets that cause the host to stop accepting connections or even crashing it. A packet filter, which is built into a robust TCP/IP stack implementation, as it is the case with Linux, can very effectively prevent such attacks or at least mitigate the effects (see Sect. 4.1).

At the application level, the either the application itself must be kept secure (i.e. not susceptible to DoS attacks) or some kind of filter which prevents maliciously-looking requests to be handed to the application has to be put in front of it.

2.3 Spoofing

With this type of attack, the attacker usually pretends to be someone else, e.g. by forging the source address in the IP header. This way access controls can be circumvented.

To protect against this kind of attack validation of the authenticity of packets is required. This is only easily possible at the network layer for certain types of spoofing (see Sect. 4.1 for implementation details).

2.4 Eavesdropping

Eavesdropping means a host "listens" to and captures data that is not addressed to it. This is especially easy on broadcast networks like Ethernet. A firewall can not be used to prevent this attack; enforcing data encryption however does, obviously.

3 Building a `netfilter` Firewall

In this section, first a short overview over the Linux `netfilter` framework architecture will be given, enabling the reader to understand the firewall scripts then built up step-by-step.

3.1 The `netfilter` Architecture

3.1.1 Basic Concepts

There are a few basic building blocks of `netfilter`: tables, chains and rules. It is essential to understand the distinction between them.

A table is serves a specific purpose. By default, there are the `filter`, `nat` and `mangle` table. As one can guess, packet filtering is mostly accomplished in the `filter` table. In contrast to chains and rules, new tables can not be created from userspace.

A chain is a list of rules, which are checked subsequently for matches against the packet that is currently "traveling" through `netfilter`. If a rule matches, the packet is delivered to the target specified in the rule. This can either be another chain or a built-in target, such as `ACCEPT` (which causes the packet to be accepted by `netfilter`).

When a packet has finished traversal through a chain and would "fall off" it at the end, it either continues traversal at the chain it came from (this is the case when the chain was the target of a rule in another chain), or is subject to *chain policy*, which can to either accept or deny the packet. Note that this directly maps onto the firewall design policy of Sect. 1.2.

Each table has a set of built-in chains. The `filter` table, which is most important in the context of this paper, has the `INPUT`, `FORWARD` and `OUTPUT` chains. They are, unsurprisingly, used for packets entering the system, packets being forwarded between network interfaces and packets leaving the system, respectively.

3.1.2 The Packet Flow

The flow through the chains is illustrated in Fig. 1[4].

When the packet enters `netfilter` via a network interface, its passed through

[4]based on diagrams in [Chen 2002] and [Andreasson 2003]

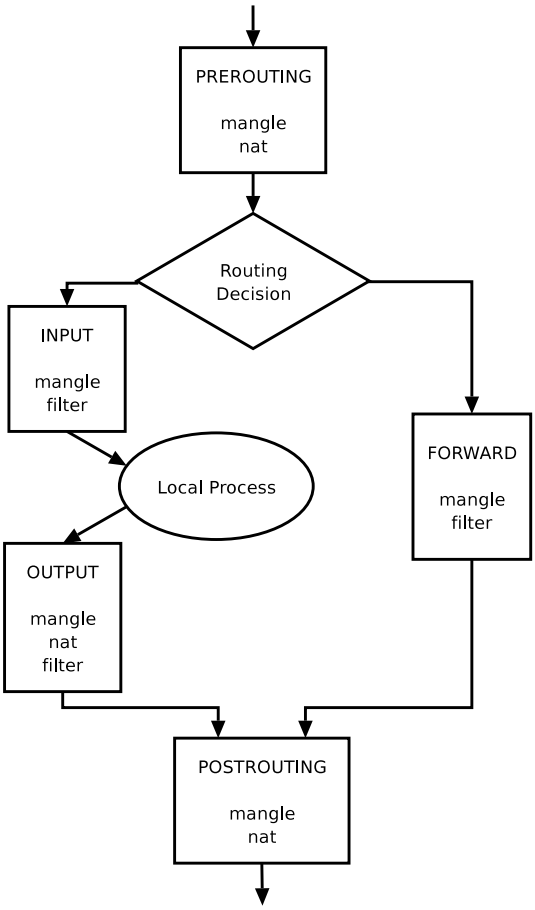

Figure 1: The packet flow through the `netfilter` chains

the PREROUTING chain of the mangle (used for manipulating the package, e.g. changing the ToS field) and nat chains, where destination NAT (like portforwarding) can be performed.

After the package has left the PREROUTING chains, the Kernel decides whether the packet is destined for the local machine; if so, the package will enter the left path in Fig. 1, where it is mangled and filtered in the INPUT chain, before being handed to a local process, given that it passes the filter and a local process is in fact willing to receive the packet; for UDP and TCP this means a process is listening on the right port and interface (address).

Packages output by a local process are mangled, NATed and filtered in the OUTPUT chain.

If the package is not destined for the local machine, it will be mangled and filtered in the FORWARD chain. Note that a packet that is forwarded never enters the INPUT or OUTPUT chains; input and output are referring to a local process, not a network interface.

After leaving the OUTPUT or FORWARD chain, the packet is once again mangled and NATed in the POSTROUTING chain. Here source NAT (like masquerading, which allows hosts in a private subnet to share an Internet connection attached to the firewall/router) is done.

3.2 Basic iptables Usage

iptables is the userspace tool for manipulating the netfilter chains in the Linux kernel. In the following, a short overview of the iptables command syntax is given. For a complete reference of available options, please refer to its manual page.

Commands that operate on entire chains:

- List a chain:
  `iptables -L` *name-of-chain*
- Create a new chain:
  `iptables -N` *name-of-chain*
- Delete an empty chain:
  `iptables -X` *name-of-chain*
- Change chain policy:
  `iptables -P` *name-of-chain* `[DROP|ACCEPT]`

- Flush the rules out of a chain:
  `iptables -F` *name-of-chain*

Commands that manipulate rules:

- Append a rule to a chain (`-A`)
- Insert a rule at a specific chain position (`-I`)
- Replace a rule at a specific position (`-R`)
- Delete a rule, either by match or position (`-D`)

To specify a table, the `-t` switch is used, defaulting to the `filter` table.

The usual way to setup a firewall is by writing a shell script that issues the necessary `iptables` commands and hook that into either into the boot process or, for allowing finer control, into the code that brings interfaces up and down. How this is done is distribution-specific; the instructions in this paper are aimed at Debian GNU/Linux, it should be not too difficult however, to adapt them for other distributions.

3.3 Firewall Design

The firewall built up in the subsequent sections has a modular design and provides the ability to be adapted to similar setups easily. Nevertheless, it follows the simple shell-script-calls-`iptables` approach; far more sophisticated solutions such as FIAIF [5] exist. The example setup assumes a firewall that is connected to the Internet and provides masqueraded Internet access for the hosts in a private LAN. The firewall hosts also some services that would be normally placed in a DMZ: DNS, Ident, Secure IMAP, SSH (for administration) and HTTP. This is not optimal from a security point of view, but keeps the setup simple (compared to a DMZ setup offering the same services).

At the core is the init script, which is hooked early into the boot process, it must run before any network interfaces are brought up, since Debian's `/etc/network/interfaces` configuration file is used for adding interface-specific configuration. By not having everything in a single script has two primary gains:

[5] `http://www.fiaif.net/`

- The approach is usable for systems with dynamic IP addresses (especially dialup systems come to mind). With dynamic IP assignment, you can not setup your rules until the interface is up and the IP is known (unless your rules don't depend on the IP).
- Modularity, which eases maintainance.

All the firewall scripts, since they qualify as configuration files, are stored in `/etc/firewall` in the example setup. The init script, since it is distribution specific and should not be needed to be changed to configure the firewall, merely is a wrapper around the scripts `common.up` and `common.down`.

The source code for `debian-init-script.sh`:

```
#!  /bin/sh
#
# File:  /etc/init.d/firewall
#
# A Debian init script for a simple iptables-based firewall.
#
# Copyright (C) 2000-2003 Andreas Rottmann <rotty@debian.org>
#

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
NAME=firewall
DESC="iptables firewall"


test -x /etc/firewall/common.up || exit 0
test -x /etc/firewall/common.down || exit 0

set -e

case "$1" in
  start)
        echo -n "Starting $DESC: $NAME"
        /etc/firewall/common.up
        echo "."
        ;;
  stop)
        echo -n "Stopping $DESC: $NAME"
        /etc/firewall/common.down
        echo "."
        ;;
  restart|force-reload)
        N=/etc/init.d/$NAME
        $N stop
        $N start
        ;;
  *)
        N=/etc/init.d/$NAME
        echo "Usage:  $N {start|stop|restart|force-reload}" >&2
        exit 1
        ;;
esac
```

```
exit 0
```

3.4 The common scripts

`common.up` first sources `/etc/firewall/options`, which contains only the path to the `iptables` utility:

```
IPTABLES=/sbin/iptables
```

It then loads some `netfilter` modules, turns on syncookie support (see Sect. 4.1), enables forwarding and sets the policies of the default chains to `DROP` (also see Sect. 1.2). At this point, the machine is completely closed and will react no kind of network traffic.

It then goes on to create a few general-purpose chains:

- `log_drop`, which simply logs a package and then drops it. This is just defined as a convenience.
- `allowed`, which accepts new connection and packages for already established connections. For classifying the packages as belonging to already established connections, the connection tracking feature of `netfilter` is used, which is described in detail in [Stephens 2002].
- `local_drop` drops packages that are reserved for private networks and should not be routed on the Internet. This could be used as spoofing protection, if there are problems with `rp_filter` (see Sect. 4.1).
- `wintraffic_drop` discards TCP and UDP traffic coming or destined to the Windows network (SMB) ports.

It then prepares the `INPUT` and `OUTPUT` chains to:

- accept traffic on the loopback device. It is critical to allow this, since otherwise connections between local processes would be blocked;
- Unconditionally enter the `input_filter` and `output_filter` chains (respectively), which are empty at the start. The interfaces-specific scripts will place their rules in these chains, so the following logging rule will trigger for all packets which are not accepted or denied explicitly in those rules.

- Log the packet.

The FORWARD chain is set up similarly, except that no rule for loopback traffic is needed, since it will never enter the FORWARD chain (consider Fig. 1). Instead of that rule, we accept already established connections.

The source code for common.up:

```
#!/bin/sh
#

LOCALHOST_IP="127.0.0.1/32"

. /etc/firewall/options

#########
# Load all required IPTables modules
#

#
# Adds some iptables targets like LOG, REJECT and MASQUARADE.
#
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_REJECT
/sbin/modprobe ipt_MASQUERADE

#
# Support for owner matching
#
#/sbin/modprobe ipt_owner

#
# Support for connection tracking of FTP and IRC.
#
/sbin/modprobe ip_conntrack_ftp
/sbin/modprobe ip_conntrack_irc

# Enable syncookies
#
echo 1 > /proc/sys/net/ipv4/tcp_syncookies

# CRITICAL: Enable IP forwarding since it is disabled by default.
#
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
# option.  This enables dynamic-ip address hacking in IP MASQ,
# making the connection with Diald and similar programs much easier.
#
#echo "1" > /proc/sys/net/ipv4/ip_dynaddr

#
# set default policies for the INPUT, FORWARD and OUTPUT chains
#
```

```
$IPTABLES -P INPUT DROP
$IPTABLES -P OUTPUT DROP
$IPTABLES -P FORWARD DROP

#
# The log_drop chain - log the packet, then drop it
#
$IPTABLES -N log_drop
$IPTABLES -A log_drop -m limit --limit 3/minute --limit-burst 3 -j LOG --log-level DEBUG --log-prefix "IPT
packet died:  "

#
# the allowed chain for TCP connections
#
# This chain will be utilised if someone tries to connect to an allowed
# port.  If they are opening the connection, or if it's already
# established (--state matching) we ACCEPT the packages, if it's not,
# we drop it.

$IPTABLES -N allowed
$IPTABLES -A allowed -p TCP --syn -j ACCEPT
$IPTABLES -A allowed -p TCP -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A allowed -p TCP -j log_drop

# local_drop
#
$IPTABLES -N local_drop
$IPTABLES -A local_drop -s 192.168.0.0/16 -j log_drop
$IPTABLES -A local_drop -s 10.0.0.0/8 -j log_drop
$IPTABLES -A local_drop -s 172.16.0.0/12 -j log_drop

#
# The wintraffic_drop chain - Dump all windows network traffice
#
WINPORTS=137,138,139,445

$IPTABLES -N wintraffic_drop
$IPTABLES -A wintraffic_drop -p TCP -m multiport --dport $WINPORTS -j log_drop
$IPTABLES -A wintraffic_drop -p UDP -m multiport --dport $WINPORTS -j log_drop
$IPTABLES -A wintraffic_drop -p TCP -m multiport --sport $WINPORTS -j log_drop
$IPTABLES -A wintraffic_drop -p UDP -m multiport --sport $WINPORTS -j log_drop

#
# INPUT chains
#
$IPTABLES -N input_filter

$IPTABLES -A INPUT -p ALL -i lo -j ACCEPT
$IPTABLES -A INPUT -j input_filter
$IPTABLES -A INPUT -m limit --limit 3/minute --limit-burst 3 -j LOG --log-level DEBUG --log-prefix "IPT
INPUT packet died:  "

#
# OUTPUT chains
#
$IPTABLES -N output_filter

$IPTABLES -A OUTPUT -o lo -j ACCEPT
$IPTABLES -A OUTPUT -j output_filter
$IPTABLES -A OUTPUT -m limit --limit 3/minute --limit-burst 3 -j LOG --log-level DEBUG --log-prefix "IPT
OUTPUT packet died:  "
```

```
#
# FORWARD chains
#
$IPTABLES -N forward_filter

$IPTABLES -A FORWARD -j forward_filter
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A FORWARD -m limit --limit 3/minute --limit-burst 3 -j LOG --log-level DEBUG --log-prefix "IPT
FORWARD packet died:  "
```

The `common.down` script is rather simple; it tears the whole firewall down by

- Setting `INPUT`, `OUTPUT` and `FORWARD` policies to `ACCEPT`,
- Flushing and deleting all chains in the `nat` and `filter` tables.

The source code for `common.down`:

```
#!/bin/sh
#

. /etc/firewall/options

$IPTABLES -P INPUT ACCEPT
$IPTABLES -P OUTPUT ACCEPT
$IPTABLES -P FORWARD ACCEPT

$IPTABLES -t nat -F
$IPTABLES -t nat -X

$IPTABLES -F
$IPTABLES -X
```

3.5 LAN Interfaces

For private LAN(s) we want to

- redirect HTTP traffic to a transparent Squid proxy (see Sect. 4.2.1),
- do a bit sanity checking regarding interfaces and IP addresses and
- allow forwarding for packets stemming from the LAN.

The source code for `lan.up`:

```
#!/bin/sh

. /etc/firewall/options
. /etc/firewall/lan.env

# Redirect HTTP packets to squid
$IPTABLES -t nat -A PREROUTING -i $LAN_IFACE -p TCP --dport 80 -j REDIRECT --to-port 3128

# accept packets when coming from the LAN interface and addressed at
# our IP on the LAN or broadcasted.
$IPTABLES -A input_filter -i $LAN_IFACE -s $LAN_IP_RANGE -d $LAN_BCAST_ADRESS -j ACCEPT
$IPTABLES -A input_filter -i $LAN_IFACE -s $LAN_IP_RANGE -d $LAN_IP -j ACCEPT

# Accept output from local processes on the LAN interface only
# if they carry the right source address
$IPTABLES -A output_filter -o $LAN_IFACE -s $LAN_IP -d $LAN_IP_RANGE -j ACCEPT

# Allow forwarding *from* the LAN interface
$IPTABLES -A forward_filter -i $LAN_IFACE -s $LAN_IP_RANGE -j ACCEPT
```

The scripts both source `lan.env`, which contains the IP address information for the lan:

```
#
# your LAN's IP range and localhost IP. /24 means to only use the first 24
# bits of the 32 bit IP adress, giving the same as netmask 255.255.255.0
#
LAN_IP_RANGE="192.168.1.0/24"
LAN_IP="192.168.1.1/32"
LAN_BCAST_ADRESS="192.168.1.255/32"
LAN_IFACE="eth0"
```

3.6 Internet Interface

Now to the most interesting part: protection against Internet traffic.

The example setup

- allows most ICMP messages,
- opens the ports for the services running on the firewall (see 3.3).
- drops all packets that arrive on the Internet interface and come from non-routed addresses,
- enables masquerading, so the hosts in the LAN can share the Internet connection;

- sets up portforwarding to some hosts in the LAN, so they can run software that accepts connections (file-sharing software comes to mind here) when it is configured to use the assigned port range;
- discards all Windows traffic that would be routed from the LAN into the Internet.

The source code for `internet.up`:

```
#!/bin/sh
#

. /etc/firewall/options
. /etc/firewall/internet.env

#
# ICMP rules
#
$IPTABLES -N icmp_packets
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type echo-request -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type echo-reply -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 3 -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 5 -j ACCEPT
$IPTABLES -A icmp_packets -p ICMP -s 0/0 --icmp-type 11 -j ACCEPT

#
# TCP ports
#
$IPTABLES -N tcp_packets
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport ssh -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport http -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport ident -j allowed
$IPTABLES -A tcp_packets -p TCP -s 0/0 --dport imaps -j allowed

#
# UDP ports
#
$IPTABLES -N udp_packets
$IPTABLES -A udp_packets -p UDP -s 0/0 --source-port domain -j ACCEPT
$IPTABLES -A udp_packets -p UDP -s 0/0 --dport domain -j ACCEPT

# Filter obviously spoofed packages
$IPTABLES -t nat -A PREROUTING -i $INET_IFACE -j local_drop

# Enable masquerading
$IPTABLES -t nat -A POSTROUTING -o $INET_IFACE -j MASQUERADE

# Portforward the 9000:9999 port range split up on the individual hosts;
# The host with the address ending in .XY gets the port range 9XY0-9XY9.

for hostpart in 2; do
  host="192.168.1."${hostpart}
  tens=$(((hostpart / 10)   ones=$((hostpart   dport_range=9${tens}${ones}0:9${tens}${ones}9
  $IPTABLES -t nat -A PREROUTING -i $INET_IFACE -p TCP \
      -d $PUBLIC_IP --dport $dport_range -j DNAT --to $host
  $IPTABLES -t nat -A PREROUTING -i $INET_IFACE -p UDP \
```

```
      -d $PUBLIC_IP --dport $dport_range -j DNAT --to $host
done

# Allow connections in the 9000:9999 port range

$IPTABLES -A forward_filter -i $INET_IFACE -p TCP --dport 9000:9999 \
    -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
$IPTABLES -A forward_filter -i $INET_IFACE -p UDP --dport 9000:9999 \
    -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT

# Dispatch the packets onto the correct chains defined above
$IPTABLES -A input_filter -p ICMP -i $INET_IFACE -j icmp_packets
$IPTABLES -A input_filter -p TCP -i $INET_IFACE -j tcp_packets
$IPTABLES -A input_filter -p UDP -i $INET_IFACE -j udp_packets

# Accept packets on already established connections
$IPTABLES -A input_filter -p ALL -i $INET_IFACE -d $PUBLIC_IP \
    -m state --state ESTABLISHED,RELATED -j ACCEPT


# Allow outgoing traffic
$IPTABLES -A output_filter -o $INET_IFACE -s $PUBLIC_IP -j ACCEPT

# Don't forward windows traffic to the Net
$IPTABLES -A forward_filter -o $INET_IFACE -j wintraffic_drop
$IPTABLES -A forward_filter -i $INET_IFACE -j wintraffic_drop
```

4 Additional Security Measures

A mere packet filter, as described above is only the basis for a firewall. For enhanced security, additional measures should be taken.

4.1 Network Security Settings

Although not related with firewall rulesets, there are some additional settings that affect network security. They can be tweaked by manipulating the settings under `/proc/sys/net/ipv4`. Settings include handling of ICMP requests, spoofing protection (`rp_filter`) and protection against SYN flood DoS attacks (Syncookies). For more information, please refer to [Lechnyr 2002].

4.2 Filtering Application Level Gateways

The firewall rulesets above restrict network traffic mostly on a IP and port basis. For more extensive filtering, application level gateways have to be used.

4.2.1 HTTP

Normally, application level gateways are not transparent. It is, however, possible to set up a transparent HTTP proxy using the Squid Web Cache [Squid]. There is a mini-HOWTO on this topic [6].

Once the HTTP traffic is running over of a proxy (transparent or not), it is possible to do advanced filtering in the proxy. For Squid there is SquidGuard [7] and squid-vscan [8].

4.2.2 E-Mail

For email, many filtering solutions are available. Centralized filtering is best realized inside a local mailer daemon, so all mail that runs through the daemon is checked. A nice for Virus and SPAM filtering using Exim 4 [9] is described in [Jackson 2003].

5 Summary

The example firewall implemented throughout Section 3 provides the basis for securing a private LAN. The solution should be flexible enough to be adapted to different setups quite easily. Setups may be, for instance:

- A single Workstation with direct Internet connectivity ("personal firewall")
- A more complicated network topology, perhaps involving a DMZ (demilitarized zone).

It is clear that a complete discussion of all relevant issues of `netfilter`-based firewalls is outside the scope of this paper.

[6] `http://en.tldp.org/HOWTO/TransparentProxy.html`
[7] `http://www.squidguard.org/`
[8] `http://www.openantivirus.org/projects.php`
[9] `http://www.exim.org`

References

[Wack et all. 1995] Keeping Your Site Comfortably Secure: An Introduction to Internet Firewalls;
`http://csrc.nist.gov/publications/nistpubs/800-10/main.html`

[Squid] Squid Web Proxy Cache Homepage;
`http://www.squid-cache.org/`

[Lechnyr 2002] Linux Gazette, Issue 77: Network Security with /proc/sys/net/ipv4;
`http://www.linuxgazette.com/issue77/lechnyr.html`

[Andreasson 2003] IPTables Tutorial;
`http://iptables-tutorial.frozentux.net/iptables-tutorial.html`

[Chen 2002] Firewalling with Netfilter/Iptables;
`http://www.knowplace.org/netfilter/index.html`

[Stephens 2002] IPTables: ConnectionTracking;
`http://www.sns.ias.edu/~jns/security/iptables/iptables_conntrack.html`

[Jackson 2003] Spam and Virus Scanning with Exim 4;
`http://www.timj.co.uk/linux/Exim-SpamAndVirusScanning.pdf`