
# TCP wrappers and IP filtering (UKERNA security workshop)

George Ross
gdmr@dcs.ed.ac.uk

November 22nd 2000


This talk is based on our experiences using wrappers and filters to protect UNIX machines from network attacks.

We are a typical (?!) biggish University department, with no particular obvious attractions. We're scanned a dozen or so times/week, with lots of other one-off probes.

Talk isn't a comprehensive tutorial on wrappers or filters (see the "howto" documentation) — that would take much longer than the time available.


## Why use wrappers and filters?

- Nothing else between you and the outside world.
- Insufficient filtering done by your provider, or you just don't trust it.
- Security in depth — in case firewall is breached, or to stop traffic which it's configured to let through.
- Alien networks (e.g. laptops at conferences or other institutions), where network is completely unknown.
- ISP dialups and cable modems are frequently scanned.


# What are TCP wrappers?

Originally (early 1990s) a small program "wrapped around" service daemons started out of inetd.

Wrapper library can also be linked in to standalone daemons.

See `http://www.porcupine.org/`

Control access based primarily on caller's source

Small and simple to configure

Logging through syslog


For daemons started by inetd, adding a wrapper is just a small change to the configuration file:

```
telnet stream tcp nowait root /usr/sbin/tcpd in.telnetd
smtp stream tcp nowait root /usr/sbin/tcpd sendmail -bs
```

The corresponding wrapper configuration ("`hosts.allow`") file:

```
in.telnetd : ALL EXCEPT .dcs.ed.ac.uk \
                    EXCEPT @external_connect
sendmail : localhost
sshd : ALL EXCEPT .dcs.ed.ac.uk EXCEPT @external_connect
sshdfwd-X11 : localhost remote.dcs.ed.ac.uk
```

The basic `hosts.allow` or `hosts.deny` file has lines containing:

- the name of the wrapped daemon, or ALL
- a host pattern, made up of: hostnames, entire domains, addresses and subnets, netgroups, ALL, EXCEPT, KNOWN or UNKNOWN, PARANOID.
- an optional shell command to run in the background

Optionally, if enabled at compile time, the shell command is replaced by extended processing options, including:

- spawn, to run a shell command in the background as before
- twist, to run a different daemon
- banners, to display a banner before calling (or not) the daemon
- setenv, to modify the daemon's environment
- user, to change the user and/or group ID of the daemon


Pro wrappers:

- lightweight and unobtrusive
- very easy to set up
- no code change for basic functionality
- can be linked into non-inetd code
- run in userspace.

Con wrappers:

- can only protect some types of service
- internal-to-inetd services aren't covered
- non-inetd services have to have library linked in
- generally only on first connection
- kernel doesn't pass everything up to wrappers.
- can't prevent bug or misfeature exploits


IP filter – it's a kernel module for Solaris, *BSD, HPUX, and IRIX. See

`http://coombs.anu.edu.au/~avalon/ip-filter.html`

It allows for all IP traffic to be filtered on one or more of:

- source address or range (using a netmask)
- destination address or range
- specific source port or source port range
- specific destination port or port range
- network interface
- IP protocol
- TCP flags
- IP options
- ICMP type

Configuration is done through a filter description language.

Logging: a userspace daemon forwards to syslog.


Separate rules for packets "in" to and "out" from the machine.

In general packets pass through all the rules, with the pass/block state being adjusted by matching rules. The "quick" keyword shortcircuits ruleset processing.

"Groups" (essentially if/then blocks) can be used to cut down on rule processing for efficiency.

Keeping of state allows outgoing packets to open a temporary hole for appropriate responses.

There are built-in proxies for "difficult" protocols (e.g. ftp).

NAT if you need it.

Choice of response (ICMP, RST, nothing) when packets are blocked.


```
...
# Group head first -- all service-wire traffic comes through here
block in proto tcp/udp from any to 129.215.216.0/24 head 216
# remote access telnet
pass in quick proto tcp from any to 129.215.216.239 port = 23 group 216
# SMTP to the mail hub
pass in quick proto tcp from any to 129.215.216.15 port = 25 group 216
# User-ftp and anonymous ftp
pass in quick proto tcp from any to 129.215.216.15 port = 21 group 216
pass in quick proto tcp from any to 129.215.216.238 port = 21 group 216
# Print servers, from EdLAN only
pass in quick proto tcp from 129.215.0.0/16 to 129.215.216.13 port = 515 group 216
pass in quick proto tcp from 129.215.0.0/16 to 129.215.216.14 port = 515 group 216
...
```

(Our boundary filters currently have 225 rules in 11 groups. The “straight-through” path has 127 rules.)


Our experience with IPfilter:

- some initial user resistance, but now generally accepted; we tighten the screw gradually
- load generally negligible with careful ruleset organisation — place most-hit rules early, use groups
- easy to build and install, but...
- loading third-party kernel modules can cause problems with manufacturers' support hotlines
- desktop and special-purpose server rulesets are *reasonably* easy to set up
- routers and general-purpose machines can be complicated and tricky to get right
- asymmetric routing negates some useful features
- FTP needs a bigger hole than is really desirable
- may need to nail down DNS ports
- daemons on non-fixed ports are awkward, but moving the anonymous-port base after they've started (`ndd` on Solaris) can help
- good log-reduction tools are *vital* — 10K-line log files are hard to read!
- *still* can't prevent bug or misfeature exploits


Or, if for Linux, there's ipchains (see `http://netfilter.samba.org/ipchains/`) which comes with 2.2.x kernels (2.3+ have netfilter instead). This does basically the same job as IPfilter, though the details are different.

The built-in chains ("INPUT," "OUTPUT" and "FORWARD") can have a default policy for packets which don't match any rules, though making it an explicit rule has advantages.

Packets start on a built-in chain. A match results in one of ACCEPT, REJECT (drop with ICMP response), DENY (drop with no reply), RETURN (from chain), or a call to a user-defined chain.

Processing of connection state is a userspace addon, not built-in. Likewise ftp handling.

Can manipulate TOS flags (for routing or network drivers).

Logging: through klogd to kernel syslog.


```
ipchains -F budp
ipchains -A budp -p udp --dport 520 -j ACCEPT
ipchains -A budp -j DENY

ipchains -F bcast
    # Broadcasts directed to running interfaces:
    # eth0...
    ipchains -A bcast -i eth0 -p udp \
        -s 129.215.58.4/255.255.255.0 -d 129.215.58.255 40: -j budp
    ipchains -A bcast -d 129.215.58.255 -j DENY
ipchains -A bcast -p udp -d 255.255.255.255 40: -j budp
ipchains -A bcast -d 255.255.255.255 -j DENY
ipchains -A bcast -d 224.0.0.0/8 -j ACCEPT
...
ipchains -P input DENY
ipchains -F input
ipchains -A input -p udp --dport 123 -j ACCEPT
ipchains -A input -i lo -j ACCEPT
ipchains -A input -s 127.0.0.0/8 -j DENY -l
ipchains -A input -d 127.0.0.0/8 -j DENY -l
ipchains -A input -j dcs
ipchains -A input -j bcast
ipchains -A input -p tcp -j tcp
ipchains -A input -p udp -j udp
ipchains -A input -p icmp \! -f -j icmp
ipchains -A input -j DENY -l
```


IPfilter vs. ipchains:

- Different OSs!
- IPfilter: kernel add-on with some OSs, built in with *BSD, more flexible filtering options, state built-in, cumulative packet disposition or "quick", groups like "if ... then".
- ipchains: comes with Linux 2.2.x, cumulative effect through chaining, chains akin to procedure calls.
- But essentially they both do more or less the same job. Just choose the one which you find easiest to support.

Wrappers are:

- *much* simpler to set up — just basic sysadmin knowledge
- lightweight, but don't see everything
- inetd-started only, unless explicitly linked in
- only affect inbound connections on end systems
- portable between systems

Filters are:

- more complex to set up, though good supporting documentation — networking expertise is useful
- see everthing, control all packets, but some overhead on all traffic
- use on end systems or on routers as basic firewall
- somewhat system-dependent

*Neither* is a panacea. They're not mutually exclusive. Use them both, but don't neglect other precautions.


TCP wrappers:

`http://www.porcupine.org/`

IPfilter:

`http://coombs.anu.edu.au/~avalon/ip-filter.html`
`http://www.obfuscation.org/ipf/`

ipchains:

`http://netfilter.samba.org/ipchains/`
`http://www.linuxdoc.org/`