Attachment 1 Version 6.3 of the 66TCP/IP daemon wrapper package.

Table of contents
---

1 - Introduction

With this package you can monitor and filter incoming requests for the SYSTAT, FINGER, FTP, TELNET, RLOGIN, RSH, EXEC, TFTP, TALK, and other network services.

It supports both 4.3BSD-style sockets and System V.4-style TLI. Praise yourself lucky if you don't know what that means.

The package provides tiny daemon wrapper programs that can be installed without any changes to existing software or to existing configuration files. The wrappers report the name of the remote host and of the requested service; the wrappers do not exchange information with the remote client process, and impose no overhead on the actual communication between the client and server applications.

Optional features are: access control to restrict what systems can connect to your network daemons; remote user name lookups with the RFC 931 protocol; additional protection against hosts that pretend to have someone elses host name; additional protection against hosts that pretend to have someone elses host address.

Early versions of the programs were tested with Ultrix >= 2.2, with SunOS >= 3.4 and ISC 2.2. Later versions have been installed on a wide variety of platforms such as SunOS 4.x and 5.x, Ultrix 3.x and 4.x, DEC OSF/1 T1.2-2, HP-UX 8.x, AIX 3.1.5 up to 3.2.4, Apollo SR10.3.5, Sony, NeXT, SCO UNIX, DG/UX, Cray, Dynix, and an unknown number of other ones.

Requirements are that the network daemons are spawned by a super server such as the inetd; a 4.3BSD-style socket programming interface and/or System V.4-style TLI programming interface; and the availability of a syslog(3) library and of a syslogd(8) daemon. The wrappers should run without modification on any system that satisfies these requirements. Workarounds have been implemented for several common bugs in systems software.

What to do if this is your first encounter with the wrapper programs: 1) read the tutorial sections for an introduction to the relevant concepts and terminology; 2) glance over the security feature sections in this document; 3) follow the installation instructions (easy or advanced). I recommend that you first use the default security feature settings. Run the wrappers for a few days to become familiar with their logs, before doing anything drastic such as cutting off access or installing booby traps.

2 - Disclaimer

The wrapper programs rely on source address information obtained from network packets. Such information is not 100 percent reliable, although the wrappers do their best to expose forgeries.

In the absence of cryptographic protection of message contents, and of cryptographic authentication of message originators, all data from the network should be treated with sound scepticism.

THIS RESTRICTION IS BY NO MEANS SPECIFIC TO THE TCP/IP PROTOCOLS.

3 - Tutorials

The tutorial sections give a gentle introduction to the operation of the wrapper programs, and introduce some of the terminology that is used in the remainder of the document: client, server, the inetd and

syslogd daemons, and their configuration files.

3.1 - How it works

Almost every application of the TCP/IP protocols is based on a client-server model. For example, when a user invokes the telnet command to connect to one of your systems, a telnet server process is executed on the target host. The telnet server process connects the user to a login process. A few examples of client and server programs are shown in the table below:

client	server	application
telnet	telnetd	remote login
ftp	ftpd	file transfer
finger	fingerd	show users

The usual approach is to run one single daemon process that waits for all kinds of incoming network connections. Whenever a connection is established, this daemon (usually called inetd) runs the appropriate server program and goes back to sleep, waiting for other connections.

The wrapper programs rely on a simple, but powerful mechanism. Instead of directly running the desired server program, the inetd is tricked into running a small wrapper program. The wrapper logs the remote host name or address and performs some additional checks. When all is well, the wrapper executes the desired server program and goes away.

The wrapper programs have no interaction with the remote user (or client process). This has two major advantages: 1) the wrappers are application-independent, so that the same program can protect many kinds of network services; 2) no interaction also means that the wrappers are invisible from outside (at least for authorized users).

Another important property is that the wrapper programs are active only when the initial contact between client and server is established. Once a wrapper has done its work there is no overhead on the client-server communication.

The simple mechanism has one major drawback: since the wrappers go away after the initial contact between client and server processes, the wrappers are of little use with network daemons that service more than one client. The wrappers would only see the first client attempt to contact such a server. The NFS mount daemon is a typical example of a daemon that services requests from multiple clients. See the section on related software for ways to deal with such programs.

There are two ways to use the wrapper programs:

1) The easy way: move network daemons to some other directory and fill the resulting holes with copies of the wrapper programs. This approach involves no changes to system configuration files, so there is very little risk of breaking things.

2) The advanced way: leave the network daemons alone and modify the inetd configuration file. For example, an entry such as:

```
tftp  dgram  udp  wait  root  /usr/etc/tcpd  in.tftpd -s /tftpboot
```

   When a tftp request arrives, inetd will run the wrapper program (tcpd) with a process name `in.tftpd'. This is the name that the wrapper will use when logging the request and when scanning the optional access control tables. `in.tftpd' is also the name of the server program that the wrapper will attempt to run when all is well. Any arguments (`-s /tftpboot' in this particular example) are

transparently passed on to the server program.

For an account of the history of the wrapper programs, with real-life examples, see the section below on related documents.

3.2 - Where the logging information goes

The wrapper programs send their logging information to the syslog daemon (syslogd). The disposition of the wrapper logs is determined by the syslog configuration file (usually /etc/syslog.conf). Messages are written to files, to the console, or are forwarded to a @loghost.

Older syslog implementations (still found on Ultrix systems) only support priority levels ranging from 9 (debug-level messages) to 0 (alerts). All logging information of the same priority level (or more urgent) is written to the same destination. In the syslog.conf file, priority levels are specified in numerical form. For example,

```
8/usr/spool/mqueue/syslog
```

causes all messages with priority 8 (informational messages), and anything that is more urgent, to be appended to the file /usr/spool/mqueue/syslog.

Newer syslog implementations support message classes in addition to priority levels. Examples of message classes are: mail, daemon, auth and news. In the syslog.conf file, priority levels are specified with symbolic names: debug, info, notice, ..., emerg. For example,

```
mail.debug


/var/log/syslog
```

causes all messages of class mail with priority debug (or more urgent) to be appended to the /var/log/syslog file.

By default, the wrapper logs go to the same place as the transaction logs of the sendmail daemon. The disposition can be changed by editing the Makefile and/or the syslog.conf file. Send a `kill -HUP' to the syslogd after changing its configuration file. Remember that syslogd, just like sendmail, insists on one or more TABs between the left-hand side and the right-hand side expressions in its configuration file.

4 - Features

4.1 - Access control

When compiled with -DHOSTS_ACCESS, the wrapper programs support a simple form of access control. Access can be controlled per host, per service, or combinations thereof. The software provides hooks for the execution of shell commands when an access control rule fires; this feature may be used to install "booby traps". For details, see the hosts_access.5 manual page, which is in `nroff -man' format. A later section describes how you can test your access control rules. The hosts_options.5 manual page describes additional features that are turned off by default.

Access control is enabled by default. It can be turned off by editing the Makefile, or by providing no access control tables. The install instructions below describe the Makefile editing process.

On System V when TCP/IP or UDP/IP is used underneath TLI, the wrapper

programs provide the same functions as with socket-based applications. When some other protocol is used underneath TLI, the host address will be some universal magic cookie that may not be usable for access control purposes.

4.2 - Host name spoofing

With some network applications, such as RSH or RLOGIN, the remote host name plays an important role in the authentication process. Host name information can be reliable when lookups are done from a _local_ hosts table, provided that the client IP address can be trusted.

With _distributed_ name services, authentication schemes that rely on host names become more problematic. The security of your system now may depend on some far-away DNS (domain name server) outside your own control.

The wrapper programs verify the remote host name that is returned by the address->name DNS server, by asking for a second opinion. To this end, the programs look at the name and addresses that are returned by the name->address DNS server.

If any name or address discrepancies are found, or if the second DNS opinion is not available, the wrappers assume that one of the two name servers is lying, and assume that the client host pretends to have someone elses host name.

When the sources are compiled with -DPARANOID, the wrappers will drop the connection in case of a host name/address discrepancy.

When the sources are not compiled with -DPARANOID, the wrappers just pretend that an offending host name is unknown.

Paranoid mode is enabled by default. It can be turned off by editing the Makefile. The configuration and installation below describes the Makefile editing process.

4.3 - Host address spoofing

While host name spoofing can be found out by asking a second opinion, it is much harder to find out that a host claims to have someone elses network address. And since host names are deduced from network addresses, address spoofing is at least as effective as name spoofing.

The wrapper programs can give additional protection against hosts that claim to have an address that lies outside their own network. For example, some far-away host that claims to be a trusted host within your own network. Such things are possible even while the impersonated system is up and running.

This additional protection is not an invention of my own; it has been present for at least five years in the BSD rsh and rlogin daemons. Unfortunately, that feature was added *after* 4.3 BSD came out, so that very few, if any, UNIX vendors have adopted it. Our site, and many other ones, has been running these enhanced daemons for several years, and without any ill effects.

When the programs are compiled with -DKILL_IP_OPTIONS, source routing will be disabled for all TCP connections that are handled by the wrapper programs.

If you are going to use this feature on SunOS 4.1.x you should apply patch 100804-03 or later. Otherwise you may experience "BAD TRAP" and "Data fault" panics when the getsockopt() system call is executed after

a TCP RESET has been received.

The feature is disabled by default. It can be turned on by editing the Makefile. The configuration and installation section below describes the Makefile editing process.

UDP services do not benefit from this additional protection. With UDP, all you can be certain of is the network packet's destination address.

4.4 - Remote username lookups

The protocol proposed in RFC 931 provides a means to get the remote user name from the client host. The requirement is that the client host runs an RFC 931-compliant daemon. The information provided by such a daemon is not intended to be used for authentication purposes, but it can provide additional information about the owner of a TCP connection.

There are some limitations: the number of hosts that run an RFC 931 (or compatible) daemon is small (but growing); remote user name lookups do not work for datagram (UDP) connections. More seriously, remote user name lookups can cause noticeable delays with connections from non-UNIX PCs. The wrappers use a 10-second timeout for RFC931 lookups, to accommodate slow networks and slow hosts.

By default, the wrappers will do username lookup only when the access control rules require them to do so. The wrappers can be configured to always perform remote username lookups by editing the Makefile. The remote username lookup timeout period (10 seconds default) can also be changed by editing the Makefile. The installation sections below describe the Makefile editing process.

The RFC 931 protocol has diverged into different directions (IDENT and TAP). To add to the confusion, both protocols use the same network port. The daemon wrappers implement a common subset of the protocols.

On System V with TLI-based network services, remote username lookups will be possible only when the underlying network protocol is TCP/IP.

4.5 - Language extensions

The wrappers sport only a limited number of features. This is for a good reason: programs that run at high privilege levels must be easy to verify. And the smaller a program, the easier to verify. There is, however, a provision to add features.

The options.c module provides a framework for language extensions. Quite a few extensions have already been implemented; they are documented in the hosts_options.5 document, which is in `nroff -man' format. Examples: changing the severity level at which a request for service is logged; "allow" and "deny" keywords; running a customized server instead of the standard one; many others.

The language extensions are not enabled by default because they introduce an incompatible change to the access control language syntax. Instructions to enable the extensions are given in the Makefile.

5 - Other works

5.1 - Related documents

The war story behind the wrapper tools is described in:

W.Z. Venema, "TCP WRAPPER, network monitoring, access control and booby traps", UNIX Security Symposium III Proceedings (Baltimore), September 1992.

ftp.win.tue.nl:/pub/security/tcp_wrapper.ps.Z (postscript)
ftp.win.tue.nl:/pub/security/tcp_wrapper.txt.Z (flat text)

The same cracker is also described in:

W.R. Cheswick, "An Evening with Berferd, In Which a Cracker is Lured, Endured, and Studied", Proceedings of the Winter USENIX Conference (San Francisco), January 1992.

research.att.com:/dist/internet_security/berferd.ps

Discussions on internet firewalls are archived on ftp.greatcircle.com. Subscribe to the mailing list by sending a message to

majordomo@greatcircle.com

With in the body (not subject): subscribe firewalls.

5.2 - Related software

Network daemons etc. with enhanced logging capabilities can generate massive amounts of information: our 100+ workstations generate several hundred kbytes each day. egrep-based filters can help to suppress some of the noise. A more powerful tool is the Swatch monitoring system by Stephen E. Hansen and E. Todd Atkins. Swatch can process log files in real time and can associate arbitrary actions with patterns; its applications are by no means restricted to security. Swatch is available from sierra.stanford.edu, directory /pub/sources.

Socks, described in the UNIX Security III proceedings, can be used to control network traffic from hosts on an internal network, through a firewall host, to the outer world. Socks consists of a daemon that is run on the firewall host, and of a library with routines that redirect application socket calls through the firewall daemon. Socks is available from s1.gov in /pub/firewalls/socks.tar.Z.

For a modified Socks version by Ying-Da Lee (ylee@syl.dl.nec.com) try ftp.nec.com, directory /pub/security/socks.cstc.

Tcpr is a set of perl scripts by Paul Ziemba that enable you to run ftp and telnet commands across a firewall. Unlike socks it can be used with unmodified client software. Available from ftp.alantec.com, /pub/tcpr.

Versions of rshd and rlogind, modified to report the remote user name in addition to the remote host name, are available for anonymous ftp (ftp.win.tue.nl:/pub/security/logdaemon-XX.tar.Z). These programs are drop-in replacements for SunOS 4.x, Ultrix 4.x, and SunOS 5.x. This archive also contains ftpd/rexecd/login versions that support S/Key one-time passwords (tested with SunOS [45] and 44BSD).

The securelib shared library by William LeFebvre can be used to control access to network daemons that are not run under control of the inetd or that serve more than one client, such as the NFS mount daemon that runs until the machine goes down. Available from eecs.nwu.edu, file /pub/securelib.tar.

xinetd (posted to comp.sources.unix) is an inetd replacement that provides, among others, logging, username lookup and access control. However, it does not support the System V TLI services, and it is only six times as much source code as the daemon wrapper programs.

netlog from Texas A&M relies on the SunOS 4.x /dev/nit interface to passively watch all TCP and UDP network traffic on a network. The current version is net.tamu.edu:/pub/security/TAMU/netlog-1.2.tar.gz.

Where shared libraries or router-based packet filtering are not an option, an alternative portmap daemon can help to improve RPC security, in particular that of NFS and of the NIS (YP) information service. ftp.win.tue.nl:/pub/security/portmap.shar.Z was tested with SunOS 4.1.1 ... 4.1.3, Ultrix 3.0 and Ultrix 4.x, HP-UX 8.x and AIX. The protection is less effective than that of the securelib library because portmap is mostly a dictionary service. SunOS 4.x users should install the latest revision of the portmap and NIS daemons instead (patch 100482) or adopt NIS+ which has access control built in.

Source for a portable RFC 931 (TAP, IDENT)-compatible daemon by Peter Eriksson is available from ftp.lysator.liu.se:/pub/ident/servers.

Some TCP/IP implementations come without syslog library. Some come with the library but have no syslog daemon. A replacement can be found in ftp.win.tue.nl:/pub/security/surrogate-syslog.tar.Z. The fakesyslog library that comes with the nntp sources reportedly works well, too.

6 - Limitations

6.1 - Known wrapper limitations

Some UDP (and rpc/udp) daemons linger around for a while after they have serviced a request, just in case another request comes in. In the inetd configuration file these daemons are registered with the `wait' option. Only the request that started such a daemon will be seen by the wrappers. Such daemons are better protected with the securelib shared library (see: Related software).

The wrappers do not work with RPC services over TCP. These services are registered as rpc/tcp in the inetd configuration file. The only non-trivial service that is affected by this limitation is rexd, which is used by the on(1) command. This is no great loss. On most systems, rexd is less secure than a wildcard in /etc/hosts.equiv.

Some RPC requests (for example: rwall, rup, rusers) appear to come from the responding host. What happens is that the client sends its request to all portmap daemons on its network; each portmap daemon forwards the request to its own system. As far as the rwall etc. daemons know, the request comes from the local host.

Portmap and RPC (e.g. NIS and NFS) (in)security is a topic in itself. See the section in this document on related software.

6.2 - Known system software bugs

Workarounds have been implemented for several bugs in system software. They are described in the Makefile. Unfortunately, some system software bugs cannot be worked around. The result is loss of functionality.

Older ConvexOS versions come with a broken recvfrom(2) implementation. This makes it impossible for the daemon wrappers to look up the remote host address (and hence, the name) in case of UDP requests. A patch is available for ConvexOS 10.1; later releases should be OK.

With early Solaris (SunOS 5) versions, the syslog daemon will leave behind zombie process when writing to logged-in users. Workaround: increase the syslogd threshold for logging to users, or reduce the wrapper's logging severity.

On some systems, the optional RFC 931 remote username lookups may trigger a kernel bug. When a client host connects to your system, and the RFC 931 connection from your system to that client is rejected by a router, your kernel may drop all connections with that client. This is not a bug in the wrapper programs: complain to your vendor, and don't enable remote user name lookups until the bug has been fixed.

Reportedly, SunOS 4.1.1, Next 2.0a, ISC 3.0 with TCP 1.3, and AIX 3.2.2 and later are OK.

Sony News/OS 4.51, HP-UX 8-something and Ultrix 4.3 still have the bug. Reportedly, a fix for Ultrix is available ((CXO-8919).

The following procedure can be used (from outside the tue.nl domain) to find out if your kernel has the bug. From the system under test, do:

```
% ftp 131.155.70.100
```

This command attempts to make an ftp connection to our anonymous ftp server (ftp.win.tue.nl). When the connection has been established, run the following command from the same system under test, while keeping the ftp connection open:

```
% telnet 131.155.70.100 111
```

Do not forget the `111' at the end of the command. This telnet command attempts to connect to our portmap process. The telnet command should fail with: "host not reachable", or something like that. If your ftp connection gets messed up, you have the bug. If the telnet command does not fail, please let me know a.s.a.p.!

For those who care, the bug is that the BSD kernel code was not careful enough with incoming ICMP UNREACHABLE control messages (it ignored the local and remote port numbers, and therefore zapped *all* connections with the remote system). The bug is still present in the BSD NET/1 source release (1989) but apparently has been fixed in BSD NET/2 (1991).

7 - Configuration and installation

7.1 - Easy configuration and installation

The "easy" recipe requires no changes to existing software or configuration files. Basically, you move the daemons that you want to protect to a different directory and plug the resulting holes with copies of the wrapper programs.

If you don't run Ultrix, you won't need the miscd wrapper program. The miscd daemon implements among others the SYSTAT service, which produces the same output as the the WHO command.

Type `make' and follow the instructions. The Makefile comes with ready-to-use templates for many common UNIX implementations (sun, ultrix, hp-ux, irix, ...).

When the `make' succeeds the result is four executables (five in case of Ultrix).

The `try' program can be used to play with host access control tables and is described in a later section.

The `safe_finger' command should be used when you implement booby traps: it gives better protection against nasty stuff that hosts may send in response to finger probes.

The `try-from' program tests the host and username lookup code. Run it from a remote shell command (`rsh host /some/where/try-from') and it should be able to figure out from what system it is being called.

The tcpd program can be used to monitor the telnet, finger, ftp, exec, rsh, rlogin, tftp, talk, comsat and other tcp or udp services that have a one-to-one mapping onto executable files.

The tcpd program can also be used for services that are marked as rpc/udp in the inetd configuration file, but not for rpc/tcp services such as rexd. You probably do not want to run rexd anyway. On most systems it is even less secure than a wildcard in /etc/hosts.equiv.

With System V.4-style systems, the tcpd program can also handle TLI services. When TCP/IP or UDP/IP is used underneath TLI, the wrappers provide the same functions as with socket-based applications. When some other protocol is used underneath TLI, functionality will be limited (no remote username lookups, weird network address formats).

Decide which services you want to monitor. Move the corresponding vendor-provided daemon programs to the location specified by the REAL_DAEMON_DIR constant in the Makefile, and fill the holes with copies of the tcpd wrapper. That is, one copy of (or link to) the tcpd program for each service that you want to monitor. For example, to monitor the use of your finger service:

```
# mkdir REAL_DAEMON_DIR
# mv /usr/etc/in.fingerd REAL_DAEMON_DIR
# cp tcpd /usr/etc/in.fingerd
```

The example applies to SunOS 4. With other UNIX implementations the network daemons live in /usr/libexec, /usr/sbin or in /etc, or have no "in." prefix to their names, but you get the idea.

File protections: the wrapper, all files used by the wrapper, and all directories in the path leading to those files, should be accessible but not writable for unprivileged users (mode 755 or mode 555). Do not install the wrapper set-uid.

Ultrix only: If you want to monitor the SYSTAT service, move the vendor-provided miscd daemon to the location specified by the REAL_DAEMON_DIR macro in the Makefile, and install the miscd wrapper at the original miscd location.

In the absence of any access-control tables, the daemon wrappers will just maintain a record of network connections made to your system.

7.2 - Advanced configuration and installation

The advanced recipe leaves your daemon executables alone, but involves simple modifications to the inetd configuration file.

Type `make' and follow the instructions. The Makefile comes with ready-to-use templates for many common UNIX implementations (sun, ultrix, hp-ux, irix, ...).

When the `make' succeeds the result is four executables (five in case of Ultrix).

The `try' program can be used to play with host access control tables and is described in a later section.

The `try-from' program tests the host and username lookup code. Run it from a remote shell command (`rsh host /some/where/try-from') and it should be able to figure out from what system it is being called.

The `safe_finger' command should be used when you implement a booby trap: it gives better protection against nasty stuff that hosts may send in response to finger probes.

The tcpd program can be used to monitor the telnet, finger, ftp, exec, rsh, rlogin, tftp, talk, comsat and other tcp or udp services that have a one-to-one mapping onto executable files.

With System V.4-style systems, the tcpd program can also handle TLI services. When TCP/IP or UDP/IP is used underneath TLI, the wrappers provide the same functions as with socket-based applications. When some other protocol is used underneath TLI, functionality will be limited (no remote username lookups, weird network address formats).

The tcpd program can also be used for services that are marked as rpc/udp in the inetd configuration file, but not for rpc/tcp services such as rexd. You probably do not want to run rexd anyway. On most systems it is even less secure than a wildcard in /etc/hosts.equiv.

Install the tcpd command in a suitable place. Apollo UNIX users will want to install it under a different name because the name "tcpd" is already taken; a suitable name would be "frontd".

File protections: the wrapper, all files used by the wrapper, and all directories in the path leading to those files, should be accessible but not writable for unprivileged users (mode 755 or mode 555). Do not install the wrapper set-uid.

Then perform the following edits on the inetd configuration file (usually /etc/inetd.conf or /etc/inet/inetd.conf):

```
    finger  stream  tcp     nowait  nobody  /usr/etc/in.fingerd     in.fingerd
                                            ^^^^^^^^^^^^^^^^^^
```

becomes:

```
    finger  stream  tcp     nowait  nobody  /usr/etc/tcpd in.fingerd
```

```
        ^^^^^^^^^^^^^
```

Send a `kill -HUP' to the inetd process to make the change effective. Some inetd implementations require that you first disable the finger service (comment out the finger service and `kill -HUP' the inetd) before you can turn on the modified version.

The example applies to SunOS 4. With other UNIX implementations the network daemons live in /usr/libexec, /usr/sbin, or /etc, the network daemons have no "in." prefix to their names, or the username field in the inetd configuration file may be missing.

When the finger service works as expected you can perform similar changes for other network services. Do not forget the `kill -HUP'.

The miscd daemon that comes with Ultrix implements several network services. It decides what to do by looking at its process name. One of the services is systat, which is a kind of limited finger service. If you want to monitor the systat service, install the miscd wrapper in a suitable place and update the inetd configuration file:

```
    systat  stream  tcp     nowait  /suitable/place/miscd      systatd
```

Ultrix 4.3 allows you to specify a user id under which the daemon will be executed. This feature is not documented in the manual pages. Thus, the example would become:

```
    systat  stream  tcp     nowait  nobody /suitable/place/miscd    systatd
```

Older Ultrix systems still run all their network daemons as root.

In the absence of any access-control tables, the daemon wrappers will just maintain a record of network connections made to your system.

7.3 - Daemons with arbitrary path names

The above tcpd examples work fine with network daemons that live in a common directory, but sometimes that is not practical. Having soft links all over your file system is not a clean solution, either.

Instead you can specify, in the inetd configuration file, an absolute path name for the daemon process name. For example,

```
    ntalk   dgram   udp     wait    root    /usr/etc/tcpd
/usr/local/lib/ntalkd
```

When the daemon process name is an absolute path name, tcpd ignores the value of the REAL_DAEMON_DIR constant, and uses the last path component of the daemon process name for logging and for access control.

7.4 - Building and testing the access control rules

In order to support access control the wrappers must be compiled with the -DHOSTS_ACCESS option. The access control policy is given in the form of two tables (default: /etc/hosts.allow and /etc/hosts.deny). Access control is disabled when there are no access control tables, or when the tables are empty.

If you haven't used the wrappers before I recommend that you first run them a couple of days without any access control restrictions. The logfile records should give you an idea of the process names and of the host names that you will have to build into your access control rules.

The syntax of the access control rules is documented in the file hosts_access.5, which is in `nroff -man' format. This is a lengthy document, and no-one expects you to read it right away from beginning to end. Instead, after reading the introductory section, skip to the examples at the end so that you get a general idea of the language. Then you can appreciate the detailed reference sections near the beginning of the document.

The examples in the hosts_access.5 document (`nroff -man' format) show two specific types of access control policy: 1) mostly closed (only permitting access from a limited number of systems) and 2) mostly open (permitting access from everyone except a limited number of trouble makers). You will have to choose what model suits your situation best. Implementing a mixed policy should not be overly difficult either.

Optional extensions to the access control language are described in the hosts_options.5 document (`nroff -man' format).

The `try' command can be used to try out your local access control files. The command syntax is:

```
    ./try process_name hostname
    (e.g.: ./try in.tftpd localhost)

    ./try process_name address
    (e.g.: ./try in.tftpd 127.0.0.1)
```

This way you can simulate what decisions will be made, and what actions

will be taken, when hosts connect to your own system.

Note 1: `try -d' will look for hosts.{allow,deny} tables in the current working directory. This is useful for testing new rules without bothering your users.

Note 2: you cannot use the `try' command to simulate what happens when the local system connects to other hosts.

In order to find out what process name to use, just use the service and watch the process name that shows up in the logfile. Alternatively, you can look up the name from the inetd configuration file. Coming back to the tftp example in the tutorial section above:

```
    tftp  dgram  udp  wait  root  /usr/etc/tcpd  in.tftpd -s /tftpboot
```

This entry causes the inetd to run the wrapper program (tcpd) with a process name `in.tftpd'. This is the name that the wrapper will use when scanning the access control tables. Therefore, `in.tftpd' is the process name that should be given to the `try' command. On your system the actual inetd.conf entry may differ (tftpd instead of in.tftpd, and no `root' field), but you get the idea.

When you specify a host name, the `try' program will use both the host name and address. This way you can simulate the most common case where the wrappers know both the host address and the host name. The `try' program will iterate over all addresses that it can find for the given host name.

When you specify a host address instead of a host name, the `try' program will pretend that the host name is unknown, so that you can simulate what happens when the wrapper is unable to look up the remote host name.

The `try' command will report serious errors to the standard error stream (no need to tail the syslog file anymore).

7.5 - Other applications

The access control routines can easily be integrated with other programs. The hosts_access.3 manual page (`nroff -man' format) describes the external interface of the libwrap.a library.

The tcpd wrapper can even be used to control access to the smtp port. This can be useful when you suspect that someone is trying out some obscure sendmail bug, or when a remote site is misconfigured and keeps hammering your mail daemon.

In that case, sendmail should not be run as a stand-alone daemon, but it should be registered in the inetd configuration file. For example:

```
    smtp    stream  tcp     nowait  root    /usr/etc/tcpd /usr/lib/sendmail
-bs
```

You will periodically want to run sendmail to process queued-up messages. A crontab entry like:

```
    0,15,30,45 * * * * /usr/lib/sendmail -q
```

should take care of that. You cannot really prevent people from posting forged mail this way, because there are many unprotected smtp daemons on the network.

8 - Acknowledgements

Many people contributed to the evolution of the programs, by asking inspiring questions, by suggesting features or bugfixes, or by submitting source code. Nevertheless, all mistakes and bugs in the wrappers are my own.

Thanks to Brendan Kehoe (brendan@cs.widener.edu), Heimir Sverrisson (heimir@hafro.is) and Dan Bernstein (brnstnd@kramden.acf.nyu.edu) for feedback on an early release of this product. The host name/address check was suggested by John Kimball (jkimball@src.honeywell.com). Apollo's UNIX environment has some peculiar quirks: Willem-Jan Withagen (wjw@eb.ele.tue.nl), Pieter Schoenmakers (tiggr@es.ele.tue.nl) and Charles S. Fuller (fuller@wccs.psc.edu) provided assistance. Hal R. Brand (BRAND@addvax.llnl.gov) told me how to get the remote IP address in case of datagram-oriented services, and suggested the optional shell command feature. Shabbir Safdar (shabby@mentor.cc.purdue.edu) provided a first version of a much-needed manual page. Granville Boman Goza, IV (gbg@sei.cmu.edu) suggested to use the remote IP address even when the host name is available. Casper H.S. Dik (casper@fwi.uva.nl) provided additional insight into DNS spoofing techniques. The bogus daemon feature was inspired by code from Andrew Macpherson (BNR Europe Ltd). Steve Bellovin (smb@research.att.com) confirmed some of my suspicions about the darker sides of TCP/IP insecurity. Risks of automated fingers were pointed out by Borja Marcos (borjam@we.lc.ehu.es). Brad Plecs (mbp@jhuspo.ca.jhu.edu) was kind enough to try my early TLI code and to work out how DG/UX differs from Solaris.

John P. Rouillard (rouilj@cs.umb.edu) deserves special mention for his persistent, but constructive, nagging about wrong or missing things, and for trying out and discussing embryonic code or ideas.

Lat but not least, Howard Chu (hyc@hanauma.jpl.nasa.gov), Darren Reed (avalon@coombs.anu.edu.au), Icarus Sparry (I.Sparry@gdr.bath.ac.uk), Scott Schwartz (schwartz@ cs.psu.edu), John A. Kunze (jak@violet.berkeley.edu), Daniel Len Schales (dan@engr.latech.edu), Chris Turbeville <turbo@cse.uta.edu>, Paul Kranenburg <pk@cs.few.eur.nl>, Marc Boucher <marc@cam.org>, Dave Mitchell <D.Mitchell@dcs.shef.ac.uk>, and many, many others provided fixes, code fragments, or other improvements to the wrappers.

Wietse Venema (wietse@wzv.win.tue.nl)
Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513
5600 MB Eindhoven
The Netherlands