

# Scaring Crackers Away with TCP Wrapper

Adam Olson

Would it not be great if computers could be connected to a network without any regard for malicious users and other security threats? In this fairy tale world, you could focus solely on tuning your systems to function as smoothly as possible. You wouldn't have to worry about a thing while implementing that intricate backup system you conjured up in your head. It is too bad this isn't the case, but there are tools out there to increase the security of these networked computer systems.

One of these tools is called TCP Wrapper. This article will cover how to configure TCP Wrapper in such a way that crackers probing your network will be afraid to investigate any further.

TCP Wrapper can also be used to verify a client's identity, control which clients can access certain services, and display custom banners. The last item will play a large part in telling the cracker you know what they are up to.

## Where is it located?

TCP Wrapper is available via **ftp** at:

```
ftp://ftp.porcupine.org/pub/security/tcp_wrappers_7.6.tar.gz
```

For a list of other security tools written by Wietse Venema, check out:

```
ftp://ftp.porcupine.org/pub/security/index.html
```

The above link also includes a version of TCP Wrapper that supports IPv6.

## Building and Installing TCP Wrapper

I performed the following steps on a system running Solaris 2.7. The steps will be very similar on most other platforms. Place the source in your favorite directory (**/var/tmp**, for example):

```
# cd /var/tmp
Unpack the source:
# gzip -d tcp_wrappers_7.6.tar.gz
Untar the source:
# tar xvf tcp_wrappers_7.6.tar
Change directories:
# cd tcp_wrappers_7.6
```

Before running **make**, you must first uncomment the appropriate line in the **Makefile** that designates the standard location of the daemon software. On Solaris 2.7, it is line 47 of the **Makefile** and it reads:

**REAL\_DAEMON\_DIR=/usr/sbin**. Run **make** and include the correct system type (type **make** for a listing of system types):

```
# make sunos5 CC=gcc
```

If you run into any compile time errors, refer to the **README** located in the **tcp\_wrappers\_7.6** directory. The **README** is also an excellent source for any questions you may have about how TCP

# Scaring Crackers Away with TCP Wrapper

Adam Olson

Wrapper functions and how it can be used. The `tcp_wrappers_7.6` directory will now include five additional binaries: `tcpdchk`, `tcpdmatch`, `safe_finger`, `try-from`, and `tcpd`.

The following is an excerpt from the **README** that provides a brief description of each binary:

You can use the `tcpdchk` program to identify the most common problems in your wrapper and `inetd` configuration files.

With the `tcpdmatch` program you can examine how the wrapper would react to specific requests for service.

The `safe_finger` command should be used when you implement booby traps: it gives better protection against nasty stuff that remote hosts may do in response to your 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.

Finally, copy the newly compiled binaries to the following directories:

```
# cp tcpd /usr/sbin
# cp safe_finger tcpdchk tcpdmatch try-from/usr/local/bin
```

## Moving On to the Configuration

There are two ways to continue with the configuration. One way involves moving daemon binaries around on your system. The other leaves them alone, and all changes are made in the configuration file of `inetd`. I recommend the latter, because I think it is easier to manage all changes in one central place. Moving around multiple files can become messy and is more prone to error. So, fire up your favorite editor and modify `inetd.conf`. In this example, I will be filtering connections to the `telnet` daemon.

A typical `telnet` entry in `inetd.conf` will look this:

```
telnet stream tcp    nowait  root  /usr/sbin/in.telnetd  in.telnetd
```

As far as running new connections through TCP Wrapper, we are primarily concerned with the last two fields. The first defines the location of the server program to start when the new connection request arrives. The second is for including any arguments that should be passed along to the server program.

Modify the `telnet` entry above to look like the following:

```
telnet stream tcp    nowait  root  /usr/sbin/tcpd  in.telnetd
```

# Scaring Crackers Away with TCP Wrapper

Adam Olson

Note that these are the steps to follow when using Solaris. Other platforms will be similar or identical. If the location of **in.telnetd** is different from what you specified as the **REAL\_DAEMON\_DIR** in the **Makefile**, you must specify the full path. For example:

```
telnet stream tcp nowait root /usr/sbin/tcpd /usr/local/sbin/in.telnetd
```

Save your changes to **inetd.conf** and send a **HUP** signal to **inetd**.

```
# kill -HUP pid_of_inetd
```

Any requests to the **telnet** service should now be logged and by default, if reverse and forward lookups on the client's address don't match, the connection will be refused. If this isn't kosher with your security policy, comment out the **PARANOID= -DPARANOID** entry in the **Makefile** and recompile.

Try opening a **telnet** session to the host from both a client whose reverse and forward name lookups match and from a client who does not have matching name lookups. After doing this, check out the log files. Logging information is by default sent to the same location as Sendmail transactions. To change this, check out the **Makefile** and **README**.

Your log file will look something like this:

```
# tail /var/log/syslog
Jun  4 19:08:28 kt in.telnetd[20483]: connect from ny.sparc5.com
Jun  4 19:09:05 kt in.telnetd[20493]: warning: can't verify hostname: getho
stbyname(bd.sparc5.com) failed
Jun  4 19:09:05 kt in.telnetd[20493]: refused connect from 206.13.45.90
```

The first connect from **ny.sparc5.com** was successful because both name lookups returned the same value. There was a discrepancy in the values returned for the second host, so the connection was refused.

## Access Control

Now that we have connections running through **tcpd**, let's start restricting access to specific client IP addresses only. This is accomplished by creating a **/etc/hosts.allow** file and a **/etc/hosts.deny** file.

Before creating the files though, let's first go over how the access control mechanism works. After receiving the connection request, **tcpd** first examines **/etc/hosts.allow** for a line that matches the requested daemon and client address. If one is found, the client is permitted. If one is not found, the **/etc/hosts.deny** file is opened and a daemon/client match is sought. If a match is found, the client is rejected. If a match is not found, the client is permitted. Thus, if there isn't a specific rule denying the client or a catch-all rule denying anyone who isn't permitted in **/etc/hosts.allow**, the client is allowed to connect to the daemon. Make careful note of this, as you don't want to leave a service open to any untrusted hosts.

You can approach access control in two ways. Permit certain hosts and deny the rest, or deny certain hosts and permit the rest. The deciding factor really depends on the environment in which the host

# Scaring Crackers Away with TCP Wrapper

Adam Olson

resides. Whenever possible, I prefer to permit a group of hosts I have a trust relationship with and deny all others.

Create a **/etc/hosts.allow** and a **/etc/hosts.deny** file identical to the following. (Substitute the allowed addresses with trusted addresses of your own.):

```
/etc/hosts.allow:
```

```
in.telnetd: 10.1.2.3 10.4.5.6
```

```
/etc/hosts.deny:
```

```
in.telnetd: ALL
```

Now try it out! You should connect successfully from any IP address in **/etc/hosts.allow** and be denied from any IP address not in **/etc/hosts.allow**. Check out the log file as well to make sure everything looks good.

The files above are an example of permitting specific trusted hosts and denying everyone else. If you wanted to do the opposite, the access files would look like:

```
/etc/hosts.allow - empty
```

```
/etc/hosts.deny:
```

```
in.telnetd: bad_user's_address problem_host
```

This configuration would permit any host not listed in **/etc/hosts.deny**.

## Custom Banners

Before I can discuss a design of how to fool a cracker into telling you what they are up to, I must address how to create custom banners.

Creating custom banners provides a way to personalize a service or provide more information, such as the administrative contact, to the client. We will be using custom banners to display a message to the intruder that should deter them from pursuing the target host any further. As long as the service being requested is using TCP, this is easily accomplished with TCP Wrapper.

We need to first recompile the package to add support for the extended options required to display banners. In addition to custom banners, these extended options can perform a number of other tasks, such as changing the security level of an event being logged, providing additional access control syntax to consolidate rules into one file, and enabling the execution of external scripts.

To recompile with support for extended options type:

```
# make clean  
# make STYLE=-DPROCESS_OPTIONS sunos5 CC=gcc  
# cp tcpd /usr/sbin
```

Revised 2001

Page 4 of 7

# Scaring Crackers Away with TCP Wrapper

Adam Olson

```
# cp safe_finger tcpdchk tcpdmatch try-from /usr/local/bin
```

The rest is straightforward. We want to display a banner to anyone who is refused a connection to the **telnet** service. This example is using the "permit some, deny the rest" mentality.

```
/etc/hosts.allow:
```

```
in.telnetd: 10.1.2.3 10.4.5.6
```

```
/etc/hosts.deny:
```

```
in.telnetd: ALL: banners /usr/local/banners
```

We just need to do one small thing before this will work -- create the banner! The "banners" in the above line tells **tcpd** to issue a banner to the client. The path that follows is the location where the banners live. So, **/usr/local/banners** needs to be created, and a banner file with the same name as the requested daemon will hold our banner text.

Using your favorite editor again, create a **/usr/local/banners/in.telnetd** file that looks like:  
Connection from unknown system refused!

Test this by opening a **telnet** session to the host from an IP address that will fall under the ALL rule in **/etc/hosts.deny**. You should be greeted with the custom banner, and then your connection will be closed.

For information on how to configure other extended options, type:

```
# nroff -man hosts_options.5 |more -s
```

## Scaring the Cracker Away

Perhaps by now you're saying, "I want to start scaring people away!" After going over the above material, we are now ready to implement the required measures. Let's first start off with how a cracker will probe your network. A common way to gather information about an unknown network is to use a tool called **nmap** or something similar. For detailed information on **nmap** and to download a copy, visit:

<http://www.insecure.org/nmap>

Upon execution, **nmap** will send a number of different types of packets to a target host or network and will return a list of active port numbers (provided they are not blocked by a firewall, router ACL, etc). Because many services have a default port that they listen on, it is fairly easy to determine which services are active and to which exploits the host or network may be vulnerable. The cracker will then attempt to connect to one of these ports to obtain additional information, such as the server program name and version.

# Scaring Crackers Away with TCP Wrapper

Adam Olson

We want to configure TCP Wrapper to listen on multiple unused ports. Choose ports normally used for common network services that the host is not providing. This would include services like smtp (port 25), http (port 80), and pop3 (110).

Let's say we have a host that should only be providing ssh and http services. We will modify **/etc/inetd.conf** to include others services that **tcpd** will be responsible for. TCP Wrapper will listen on unused ports that crackers commonly attempt to contact (such as the **ftp** and **telnet** ports) and will send a blistering message to any intruder who attempts to contact the port. If these ports are ever contacted, there is a very high chance someone is poking around.

An example copy of **/etc/inetd.conf** on Solaris is:

```
ftp      stream  tcp      nowait  root    /usr/sbin/tcpd  in.ftpd
telnet   stream  tcp      nowait  root    /usr/sbin/tcpd  in.telnetd
```

**/etc/hosts.allow** would be empty. **/etc/hosts.deny** would look like this:

```
ALL: ALL: banners /usr/local/banners : spawn (/usr/bin/mailx -s "tcpd violation
from %h - possible cracker!" user@domain.com) &
```

Note that **%h** is expanded to include the address of the source host. For a list of other variables you might want to use, type:

```
nroff -man hosts_access.5 |more -s
```

Send a **HUP** signal to **inetd**, and you are set:

```
# kill -HUP pid_of_inetd
```

TCP Wrapper should now be listening for incoming connections on port 21 and port 23. The connection will be refused and a custom banner will be displayed to the cracker. Make sure that the correct files exist in **/usr/local/banners**, as they must match the name of the daemon.

An example custom banner for the **ftp** daemon would be located at **/usr/local/banners/in.ftpd** and would look something like:

```
This service is not provided by this host!
An administrator has been notified of this
connection attempt and your IP address has
been logged for our records!
All traffic to this host is now being captured
and logged!
```

That last line is an extra incentive for the cracker to run away. If you dont currently capture traffic that strays from your normal traffic trends, I suggest checking out:

<http://www.snort.org>

# Scaring Crackers Away with TCP Wrapper

Adam Olson

Snort is an excellent lightweight network intrusion detection program that I use to capture all "interesting" traffic 24x7. There is also an article on Snort in the September issue of *Sys Admin*.

## From the Crackers Point of View

To test this setup, pretend you are probing your own network. You will see something like the following:

```
# ./nmap hostname

Starting nmap V. 2.3BETA14 by fyodor@insecure.org ( www.insecure.org/nmap/ )
Unable to find nmap-services! Resorting to /etc/services
Interesting ports on host (10.1.1.1):

Port      State      Protocol  Service
21        open      tcp       ftp
22        open      tcp       unknown
23        open      tcp       telnet
80        open      tcp       unknown
```

```
Nmap run completed -- 1 IP address (1 host up) scanned in 0 seconds
```

```
# telnet hostname 21
Trying 10.1.1.1...
Connected to host.domain.com.
Escape character is '^]'.
```

```
This service is not provided by this host!
An administrator has been notified of this
connection attempt and your IP address has
been logged for our records!
All traffic to this host is now being captured
and logged.
```

```
Connection closed by foreign host.
```

Also, verify that an alert was received at the email address defined earlier.

## Conclusion

TCP Wrapper is an excellent tool to use for early notification of a crackers attempt to probe your network, as well as scaring them away. If this doesnt deter crackers, the administrator is still alerted to whats going on and can immediately either watch them or block them from entering the network with ingress filtering. TCP Wrapper has other uses that were not covered in this article, so please check out the **README** file to see if those interest you.