TCP WRAPPER
Network monitoring, access control, and booby traps.

Wietse Venema

Mathematics and Computing Science
Eindhoven University of Technology
The Netherlands
`wietse@wzv.win.tue.nl`

Abstract

This paper presents a simple tool to monitor and control incoming network traffic. The tool has been successfully used for shielding off systems and for detection of cracker activity. It has no impact on legal computer users, and does not require any change to existing systems software or configuration files. The tool has been installed world-wide on numerous UNIX systems without any source code change.

1. Our pet.

The story begins about two years ago. Our university was under heavy attack by a Dutch computer cracker who again and again managed to acquire `root` privilege. That alone would have been nothing more than an annoyance, but this individual was very skilled at typing the following command sequence:

```
rm -rf /
```

For those with no UNIX experience: this command, when executed at a sufficiently high privilege level (like `root`), is about as destructive as the MS-DOS `format` command. Usually, the damage could be repaired from backup tapes, but every now and then people still lost a large amount of work.

Though we did have very strong indications about the cracker´s identity I cannot disclose his name. We did give him a nickname, though: "our pet"[1].

2. The cracker is watching us.

The destructive behavior of the cracker made it very hard to find out what was going on: the `rm -rf` removed all traces very effectively. One late night I noticed that the cracker was watching us over the network. He did this by frequently making contact with our `finger` network service, which gives information about users. Services such as `finger` do not require a password, and almost never keep a record of their use. That explains why all his fingering activity had remained unnoticed.

The natural reaction would be to shut down the `finger` network service. I decided, however, that it would be more productive to maintain the service and to find out where the `finger` requests were coming from.

3. A typical UNIX TCP/IP networking implementation.

In order to explain the problem and its solution I will briefly summarize a typical UNIX implementation of the TCP/IP network services. Experts will forgive me when I make a few simplifications.

---

1. Like *hond* (dog), *kat* (cat), and *muis* (mouse).

Almost every application of the TCP/IP protocols is based on a *client-server* model. For example, when someone uses the `telnet` command to connect to a host, a *telnet server* process is started on the target host. The server process connects the user to a `login` process. A few examples are shown in table 1.

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

Table 1. Examples of TCP/IP client-server pairs and their applications.

The usual approach is to run one *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.

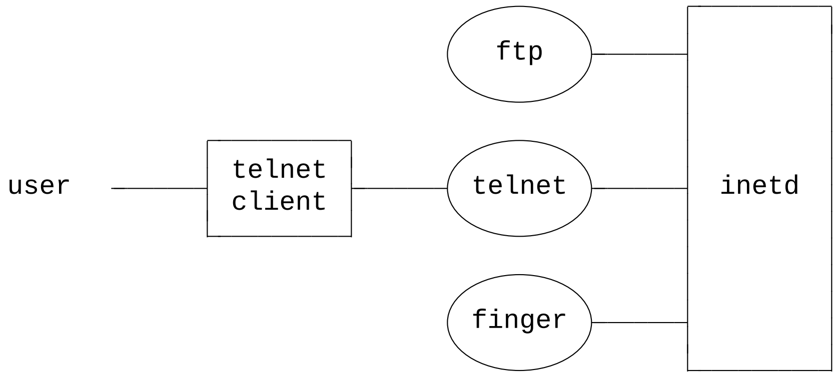

Figure 1. The `inetd` daemon process listens on the `ftp, telnet` etc. network ports and waits for incoming connections. The figure shows that a user has connected to the `telnet` port.

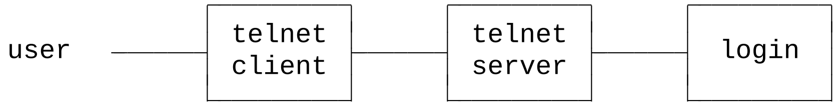

Figure 2. The `inetd` process has started a *telnet server* process that connects the user to a `login` process. Meanwhile, `inetd` waits for other incoming connections.

4. The "tcp wrapper" trick.

Back to the original problem: how to get the name of the host that the cracker was spying from. At first sight, this would require changes to existing network software. There were a few problems, though:

- o We did not have a source license for the Ultrix, SunOS and other UNIX implementations on our systems. And no, we did not have those sources either.
- o The Berkeley network sources (from which most of the commercial UNIX TCP/IP network implementations are derived) were available, but porting these to our environments would require an unknown amount of work.

Fortunately, there was a simple solution that did not require any change to existing software, and that turned out to work on all UNIX systems that I tried it on. The trick was to make a swap: move the vendor-provided network server programs to another place, and install a trivial program in the original place of the network server programs. Whenever a connection was made, the trivial program would just record the name of the remote host, and then run the original network server program.

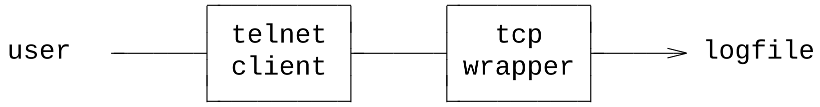

Figure 3. The original `telnet` server program has been moved to some other place, and the *tcp wrapper* has taken its place. The wrapper logs the name of the remote host to a file.

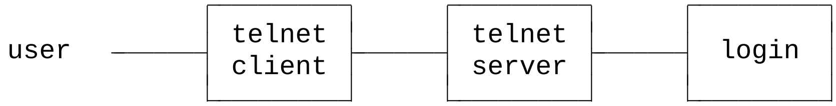

Figure 4. The *tcp wrapper* program has started the real `telnet` server and no longer participates. The user cannot notice any difference.

The first *tcp wrapper* version was just a few lines of code that I had carefully copied from some old network daemon source. Because it did not exchange any information with the client or server processes, the same *tcp wrapper* version could be used for many types of network service.

Although I could install the wrapper only on a dozen systems it was an immediate success. Figure 5 gives an early example.

```
May 21 14:06:53 tuegate: systatd: connect from monk.rutgers.edu
May 21 16:08:45 tuegate: systatd: connect from monk.rutgers.edu
May 21 16:13:58 trf.urc: systatd: connect from monk.rutgers.edu
May 21 18:38:17 tuegate: systatd: connect from ap1.eeb.ele.tue.nl
May 21 23:41:12 tuegate: systatd: connect from mcl2.utcs.utoronto.ca
May 21 23:48:14 tuegate: systatd: connect from monk.rutgers.edu

May 22 01:08:28 tuegate: systatd: connect from HAWAII-EMH1.PACOM.MIL
May 22 01:14:46 tuewsd:  fingerd: connect from HAWAII-EMH1.PACOM.MIL
May 22 01:15:32 tuewso:  fingerd: connect from HAWAII-EMH1.PACOM.MIL
May 22 01:55:46 tuegate: systatd: connect from monk.rutgers.edu
May 22 01:58:33 tuegate: systatd: connect from monk.rutgers.edu
May 22 02:00:14 tuewsd:  fingerd: connect from monk.rutgers.edu
May 22 02:14:51 tuegate: systatd: connect from RICHARKF-TCACCIS.ARMY.MIL
May 22 02:19:45 tuewsd:  fingerd: connect from RICHARKF-TCACCIS.ARMY.MIL
May 22 02:20:24 tuewso:  fingerd: connect from RICHARKF-TCACCIS.ARMY.MIL

May 22 14:43:29 tuegate: systatd: connect from monk.rutgers.edu
May 22 15:08:30 tuegate: systatd: connect from monk.rutgers.edu
May 22 15:09:19 tuewse:  fingerd: connect from monk.rutgers.edu
May 22 15:14:27 tuegate: telnetd: connect from cumbic.bmb.columbia.edu
May 22 15:23:06 tuegate: systatd: connect from cumbic.bmb.columbia.edu
May 22 15:23:56 tuewse:  fingerd: connect from cumbic.bmb.columbia.edu
```

Figure 5. Some of the first cracker connections observed with the *tcp wrapper* program. Each connection is recorded with: time stamp, the name of the local host, the name of the requested service (actually, the network server process name), and the name of the remote host. The examples show that the cracker not only used dial-up terminal servers (such as `monk.rutgers.edu`), but also that he had broken into military (`.MIL`) and university (`.EDU`) computer systems.

The cracker literally bombarded our systems with `finger` and `systat` requests. These allowed him to see who was on our systems. Every now and then he would make a `telnet` connection, presumably to make a single *login* attempt and to disconnect immediately, so that no "repeated login failure" would be reported to the systems console.

Thus, while the cracker thought he was spying on us we could from now on see where he was. This was a major improvement over the past, when we only knew something had happened after he had performed his `rm -rf` act.

My initial fear was that we would be swamped by logfile information and that there would be too much noise to find the desired signal. Fortunately, the cracker was easy to recognize:

- He often worked at night, when there is little other activity.
- He would often make a series of connections to a number of our systems. By spreading his probes he perhaps hoped to hide his activities. However, by merging the logs from several systems it was actually easier to see when the cracker was in the air.
- No-one else used the `systat` service.

In the above example, one of the `systat` connections came from a system within our university: `ap1.eeb.ele.tue.nl`, member of a ring of Apollo workstations. Attempts to alert their system administrator were in vain: one week later all their file systems were wiped out. The backups were between one and two years old, so the damage was extensive.

5. First extension: access control.

I will not go into a discussion on the pros and cons of publicly-accessible terminal servers with world-wide internet access, but it is clear that any traces that originated from such a system would be useless for our purposes: we would need cooperation from US and Dutch telephone companies, from the administrators of those terminal servers, and so on.

The best thing to do was to refuse connections from open terminal servers, so that the cracker could reach us only after breaking into a regular user account. Our hope was that the would leave some useful traces, so that we would get to know him a little better.

I built a simple access-control mechanism into the *tcp wrapper*. Whenever a connection from a terminal server showed up in the logs, all traffic from that system would be blocked on our side, and we would ask the responsible administrators to do the same on their side. Sometimes it even worked. Figure 6 gives a snapshot of our access-control files.

```
/etc/hosts.allow:

    in.ftpd: ALL

/etc/hosts.deny:

    ALL: terminus.lcs.mit.edu hilltop.rutgers.edu monk.rutgers.edu
    ALL: comserv.princeton.edu lewis-sri-gw.army.mil
    ALL: ruut.cc.ruu.nl 131.211.112.44
    ALL: tip-gsbi.stanford.edu
    ALL: tip-quada.stanford.edu
    ALL: s101-x25.stanford.edu
    ALL: tip-cdr.stanford.edu
    ALL: tip-cromemaa.stanford.edu
    ALL: tip-cromembb.stanford.edu
    ALL: tip-forsythe.stanford.edu
```

Figure 6. Sample access-control files. The first file describes which (service, host) combinations are allowed. In this example, the `ftp` file transfer service is granted to all systems. The second file describes which of the remaining (service, host) combinations are disallowed. In this example, an ever-growing list of open terminal servers is refused access. (service, host) pairs that are not matched by any of the access-control files are always allowed.

6. Our turn: watching the cracker.

Now that the cracker could no longer attack us from publicly-accessible terminal servers, all he could do was to break into a regular user account and proceed from there. That is exactly what he did. The next step was to find out what user accounts were involved.

I quickly cobbled together something that would consult a table of "bad" sites and send a `finger` and `systat` probe whenever one made a connection to us. Now we would be able to watch the cracker just like he had been watching us.

During the next months I identified several broken-into accounts. Each time I would send a notice to the system administrators, and a copy to CERT[2] to keep them informed of our progress.

```
Jan 30 04:55:09 tuegate: telnetd: connect from guzzle.Stanford.EDU
Jan 30 05:10:02 svin01:  fingerd: connect from guzzle.Stanford.EDU
Jan 30 05:17:57 svin01:  fingerd: connect from guzzle.Stanford.EDU
Jan 30 05:18:24 svin01:  fingerd: connect from guzzle.Stanford.EDU
Jan 30 05:18:34 svin01:  fingerd: connect from guzzle.Stanford.EDU
Jan 30 05:18:38 svin01:  fingerd: connect from guzzle.Stanford.EDU
Jan 30 05:18:44 svin01:  fingerd: connect from guzzle.Stanford.EDU
Jan 30 05:21:03 svin01:  fingerd: connect from guzzle.Stanford.EDU
Jan 30 05:24:46 tuegate: systatd: connect from guzzle.Stanford.EDU
Jan 30 05:27:20 svin01:  fingerd: connect from gloworm.Stanford.EDU
Jan 30 05:33:33 svin01:  telnetd: connect from guzzle.Stanford.EDU
Jan 30 05:33:38 svin01:  telnetd: connect from guzzle.Stanford.EDU
Jan 30 05:33:41 svin01:  telnetd: connect from guzzle.Stanford.EDU
Jan 30 05:33:50 svin01:  ftpd:    connect from guzzle.Stanford.EDU
Jan 30 05:33:58 svin01:  fingerd: connect from math.uchicago.edu
Jan 30 05:34:08 svin01:  fingerd: connect from math.uchicago.edu
Jan 30 05:34:54 svin01:  fingerd: connect from math.uchicago.edu
Jan 30 05:35:16 svin01:  fingerd: connect from guzzle.Stanford.EDU
Jan 30 05:35:36 svin01:  fingerd: connect from guzzle.Stanford.EDU
```

Figure 7. A burst of network activity, most of it from Stanford.

```
Wed Jan 30 05:10:08 MET 1991

[guzzle.stanford.edu]
Login name: adrian                          In real life: Adrian Cooper
Directory: /u0/adrian                       Shell: /phys/bin/tcsh
On since Jan 29 19:30:18 on ttyp0 from tip-forsythe.Sta
No Plan.
```

Figure 8. A *reverse finger* result, showing that only one user was logged on at the time.

The examples in figures 7 and 8 show activity from a single user who was logged in on the system `guzzle.Stanford.EDU`. The account name is `adrian`, and the login came in via the terminal server `tip-forsythe.Stanford.EDU`. Because of that terminal server I wasn't too optimistic. Things turned out to be otherwise.

CERT suggested that I contact Stephen Hansen of Stanford university. He had been monitoring the cracker for some time, and his logs gave an excellent insight into how the cracker operated. The cracker did not use any black magic: he knew many system software bugs, and was very persistent in his attempts to get superuser privilege. Getting into a system was just a matter of finding an account with a weak password.

---

2. Computer Emergency Response Team, an organization that was called into existence after the *Internet worm* incident in 1988.

For several months the cracker used Stanford as his home base to attack a large number of sites. One of his targets was `research.att.com`, the AT&T Bell labs gateway. Bill Cheswick and colleagues even let him in, after setting up a well-protected environment where they could watch him. This episode is extensively described in [1].

Unfortunately, the cracker was never arrested. He should have waited just one year. Instead, the honor was given to two much less harmful Dutch youngsters, at the end of February, 1992.

7. Second extension: booby traps.

Automatic reverse fingers had proven useful, so I decided to integrate the "ad hoc" reverse finger tool with the tcp wrapper. To this end, the access-control language was extended so that arbitrary shell commands could be specified.

Now that the decision to execute shell commands was based on both the service and the host name, it became possible to automatically detect some types of "suspicious" traffic. For example: remote access to network services that should be accessed only from local systems.

Over the past months I had noticed several `tftp` (trivial file transfer protocol) requests from far-away sites. This protocol does not require any password, and it is often used for downloading systems software to diskless workstations or to dedicated network hardware. Until a few years ago, the protocol could also be used to read any file on the system. For this reason, it is still popular with crackers.

The access-control tables (fig. 9) were set up such that *local* `tftp` requests would be handled in the usual manner. *Remote* `tftp` requests, however, would be refused. Instead of the requested file, a *finger* probe would be sent to the offending host.

```
/etc/hosts.allow:

    in.tftpd: LOCAL, .win.tue.nl

/etc/hosts.deny:

    in.tftpd: ALL: /usr/ucb/finger -l @%h 2>&1 | /usr/ucb/mail wswietse
```

Figure 9. Example of a *booby trap* on the `tftp` service. The entry in the first access-control file says that `tftp` connections from hosts within its own domain are allowed. The entry in the second file causes the *tcp wrapper* to perform a *reverse finger* in all other cases. The `%h` sequence is replaced by the actual remote host name. The result is sent to me by electronic mail.

The alarm goes off about once every two months. The action is as usual: send a message to CERT and to the site contact (never to the broken-into system).

This is an example of recent `tftp` activity:

```
Jan  4 18:58:28 svin02 tftpd: refused connect from E40-008-8.MIT.EDU
Jan  4 18:59:45 svin02 tftpd: refused connect from E40-008-8.MIT.EDU
Jan  4 19:01:02 svin02 tftpd: refused connect from E40-008-8.MIT.EDU
Jan  4 19:02:19 svin02 tftpd: refused connect from E40-008-8.MIT.EDU
Jan  4 19:03:36 svin02 tftpd: refused connect from E40-008-8.MIT.EDU
Jan  4 19:04:53 svin02 tftpd: refused connect from E40-008-8.MIT.EDU
```

Due to the nature of the `tftp` protocol, the refused request was repeated every 77 seconds. The retry interval is implementation dependent and can give some hints about the type of the remote system.

According to the *reverse finger* results, only one person was active at that time: apparently, the login came from a system in France.

```
Login name: mvscott                        In real life: Mark V Scott
Office: 14S-134,  x3-6724
Directory: /mit/mvscott                    Shell: /bin/csh
On since Jan  4 12:46:44 on ttyp0 from cnam.cnam.fr
12 seconds Idle Time
No Plan.
```

France told me that the cracker came from a NASA terminal server (`sdcds8.gsfc.nasa.gov`):

```
hyper1    ttyp3    sdcds8.gsfc.nasa Sat Jan  4 17:51 - 20:47  (02:55)
```

Evidently, this person liked to cross the Atlantic a lot: from NASA to France, from France to MIT, and from MIT to the Netherlands.

The example in this section gives only a limited illustration of the use of booby traps. Booby traps can be much more useful when installed on *firewall* systems [2], whose primary purpose is to separate an organizational network from the rest of the world. A typical firewall system provides only a limited collection of network services to the outer world, for example: `telnet` and `smtp`. By placing booby traps on the remaining network ports one can implement an effective early-warning system [1].

8. Conclusions.

The *tcp wrapper* is a simple but effective tool for monitoring and controlling network activity. Our FTP logs show that it has been installed in almost every part of the world, and that it is being picked up almost every day.

To briefly recapitulate the essential features of the tool:

o There is no need to modify existing software or configuration files.

o The default configuration is such that the software can be installed "out of the box" on most UNIX implementations.

o No impact on legal users.

o The wrapper program does not exchange any data with the network client or server process, so that the risk of software bugs is extremely small.

o It is suitable for both TCP (connection oriented) and UDP (datagram) services that are covered by a central daemon process such as the `inetd`.

o Protection against hosts that pretend to have someone elses name (name server spoofing). This is important for network services such as `rsh` and `rlogin` whose authentication scheme is based on host names. When a host name or address mismatch is detected the connection is dropped even before the access-control files are consulted.

o The optional access-control facility can be used to shield off open systems. Network routers can perform a similar function, but they seldom keep a record of unwanted traffic. On the other hand, network routers can be useful to block access to ports that normally cannot be covered with wrapper-like programs, such as the *portmapper*, *NIS*, *NFS* and *X server* network ports.

o The *booby-trap* facility can be used to implement early-warning systems. This can be especially useful for so-called *firewall* computer systems that only provide a limited set of network services to the outer world. The remaining network ports can be turned into booby traps.

Of course, the *tcp wrapper* is just one of the things I have set up on our systems: many other trip wires have been installed as well. Fortunately, I was able to do so before our present system administrator was installed. In any case, Dutch crackers seem to think that the systems at Eindhoven University are reasonably protected.

9. Availability.

Several releases of the *tcp wrapper* source have featured in the USENET comp.sources.misc newsgroup. The most recent version is available from:

```
ftp.uu.net:/comp.sources.misc/volumexx/log_tcp,
cert.org:/pub/tools/tcp_wrappers/tcp_wrappers.*,
ftp.win.tue.nl:/pub/security/log_tcp.shar.Z.
```

10. About the author.

Wietse Zweitze Venema studied experimental nuclear physics at Groningen University. After finishing his Ph.D. dissertation on left-right symmetry in nuclear beta decay he joined the Mathematics and Computing Science department at the Eindhoven University of Technology, where he is now a consultant at the division of Operations Research, Statistics and Systems Theory.

11. References.

[1] 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.

[2] S. Carl-Mitchell, J.S. Quarterman, *Building Internet Firewalls*. UnixWorld, February 1992.