Using tcpwrappers to save your system

(and your bacon).

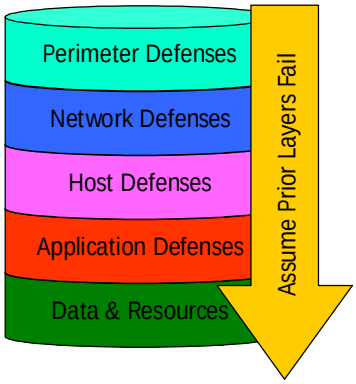
Dillon Pyron, CISSP
NETSerenity, Inc.
dillon@netserenity.net

Security Concepts

Security for any networked system must be viewed as an onion (time worn analogy, but appropriate). Rather than peeling back the layers, however, consider how many layers you must get through to reach the middle. By imposing properly designed and maintained layers of protection between "the bad guys" and your sensitive data, you can afford yourself a cascade of defense. This is called "defense-in-depth".

Defense-In-Depth

The defense in depth concept revolves around what can be viewed as a stack (the ISO model comes to mind) or a series of concentric rings, with your data or other assets protected from the attackers. And example of this view is shown below. Notice the



progressive nature of the protection in the event a previous level fails. An environment that does not employ defense-in-depth, or that does not deploy it properly, can expose itself to severe danger in the event that a layer previously considered "adequate" should fail. Such a system is frequently referred to as egg-like (hard shell, soft interior) or "crunchy on the outside, chewy on the inside".

Perimeter

Perimeter defenses are firewalls and similar enterprise wide technical assets. These must be scaled to address the broadest spectrum of both internal requirements and external threats. Typically, perimeter defenses receive the largest portion of the investment in

time and money, because of their outward facing nature and the emphasis on stopping external attackers.

Network

A network level defense is positioned inside of the primary firewall. It can consist of resources such as routers with access control lists (ACL's), which are also frequently seen on perimeters. A network defense most frequently utilizes sensors that detect either known attack signatures or anomalous network conditions, or both. Vendors provide different options on how to address the attacks, with most providing some level of tailoring. The options can range from simple logging, to breaking the connection to activation of custom scripts. Care must be taken, to avoid overwhelming the sensors and consoles with either inconsequential events or false positives.

Host

A host-based IDS is designed to detect and (hopefully) respond on attacks directed at a specific host. Typically, this is a software system running on the host that is protected, although inline, hardware solutions are also present. These hardware solutions, however, are costly and may be difficult to manage. As a result, software is the usual route taken.

Introduction to tcpwrappers

Tcpwrappers was originally developed by Weiste Venema as a means to identify, track and deal with attacks on his systems. Over the years, it has expanded in both scope and capabilities to its present status as one of the most widely used host-based IDS systems.

Tcpwrappers uses facilities already present in all Unix® and Linux® variants. It uses inetd to trap and syslogd to report and record activity. It will also use In fact, the daemon tcpd is frequently also available. However, I would recommend going to ftp://ftp.porcupine.org/pub/security to download the latest release. Make the daemon and install it in /usr/sbin.

Tcpwrappers work by interposing themselves between "the world" and a server application. For instance, a wrapper might be placed between a telnet user and the telnet daemon. However, this wrapper only exists for a short period of time. Once the initial connection is made between the client and the server (application), the wrapper is no longer involved.

The wrapper works by identifying the intended transaction and using a lookup table (provided by you) to determine the action(s) to be taken. The flexibility of this plan allows a system to allow or deny connections from certain users or locations, to log connection requests or to take other, specialized actions. As seen below, the options are almost limitless.

To put the wrappers into action, you will need to make minor modifications to /etc/inetd.conf. As an example:

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

becomes:

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

What happens here is that /usr/sbin/tcpd is called by inetd in place of the usual daemon. It is called with the argument of the daemon that will be used. When tcpd is started, it starts with the name provided as this argument. When it performs its processing and determines that the daemon should be started (this may not always be the case), the daemon provided by the argument is started.

Now, how does this "in between" daemon determine whether the ultimate daemon should be started? Two simple files are used. The format for both /etc/hosts.allow and /etc/hosts.deny are identical. Each contains a service and a hostlist that will be allowed/denied, as appropriate. The format is of the form `service:hostlist:action.` The default for each is the keyword ALL. In fact, in the absence of a file `ALL:ALL:ALLOW` is used. In the event both files exist, `hosts.allow` is read first and overrides entries in `hosts.deny`. We can tell our server to refuse access to anyone using `hosts.deny` and allow access to specific hosts via `hosts.allow.`

The action option can be as simple as an allow or deny, or a complicated action to log or take some other action. In addition to the action here, alternative daemons can be started. For instance, if you have no use for telnet (and you shouldn't), why not write a little "daemon" that spoofs the actions of telnetd, but never lets the person actually login, even with a correct username/password combination (nasty piece of work).

Examples

Now that we know how it works, lets look at how to make it work. In this example, we'll allow only one host access via ftp, nobody gets telnet, only our internal network gets ssh and we'll do some special logging with finger. First step (recommended) is to download the current version of the daemon, make it and move it to /usr/sbin. Next, we need to modify /etc/inetd.conf to handle the new daemon.

```
ftp    stream tcp nowait root /usr/sbin/tcpd  /usr/sbin/ftpd
telnet stream tcp nowait root /usr/sbin/tcpd  /usr/sbin/telnetd.in
ssh    stream tcp nowait root /usr/sbin/tcpd  /usr/sbin/sshd
finger stream tcp nowait root /usr/sbin/tcpd  /usr/sbin/fingerd
```

At this point, we create the file /etc/hosts.allow and /etc/hosts.deny.

First, in hosts.allow we put the following lines in.

```
ftp:  192.168.1.101:    ALLOW
ssh:  192.168.1.*:      ALLOW
```

Hosts.deny will look like this.

```
fingerd:    ALL:  spawn (echo Finger attempt from %h %a on `date`
|/var/log/tcp.deny.log
ALL:ALL
```

Notice the "catch all" `ALL:ALL` at the end of the file. Anything not specifically allowed will be denied. The entry for finger will spawn a process that makes an entry into the logfile tcp.deny.log, which you should be checking periodically.

HP-UX and inetd.sec

HP has developed a form of tcpwrappers for HP-UX. It uses a single file and requires no changes to inetd.conf. However, it lacks flexibility in that the only actions available are to either allow or deny access to the service. In most cases, however, this is adequate to provide that extra bit of security.

In place of /etc/hosts.allow and /etc/hosts.deny, the file /usr/adm/inetd.sec is used. Whenever inetd is invoked, it searches for the file. If it exists, then the file is searched for the service and the action to be taken is performed if the service is in the file.

The format of the file is similar to the /etc/hosts.* files.

```
<service name> <allow|deny> <host/net addresses, host/net names>
```

Examples

In this quick example, we will let anyone on the internal network get in via ssh, deny finger access to everyone and allow ftp access from only one machine.

```
#give our internal people ssh access
ssh allow 192.168.1.*
#deny finger to everybody
fingerd deny *
#ftp only from my machine (greedy :-)
ftp allow 192.168.1.101
```

Simple, although not as flexible.