In this chapter:

- *Security Resources*
- *A Handful of Security Tips*
- *Monitoring Security*
- *Boiling It All Down*

13

Addressing IMAP Security

Keeping your IMAP server secure is no different from keeping your other resources secure. Well…let's back up for a second. You can't keep your IMAP server secure any more than you can keep anything else secure. Short of sealing your server in a block of titanium and firing it off into the sun, the best you can do is keep your server *mostly* secure. The standard test of whether you're spending too much time on security is whether you've made it more difficult to compromise your system than the rewards of doing so are worth. Of course, hell-bent, disgruntled employees probably think any level of compromise is worth any amount of effort, so we'll expend a bit more effort on their behalf.

There are three things to keep in mind:

- Stay informed!
- Stay updated!
- Stay vigilant!

The best thing you can do to stay informed is to make it a habit of reviewing online resources, such as mailing lists, Usenet newsgroups, and web sites, for information about recently discovered vulnerabilities in all the various operating systems, servers, clients, and tools you use. It's safe to say that because your professional world revolves around providing service to your users and, at best, security is a secondary concern, you'll be far from the first person to learn about vulnerabilities on your system. Hackers,* on the other hand, are likely to live in a world that revolves *entirely* around discovering vulnerabilities in your system and

* The popular press has narrowed common usage until crackers, phreaks, and hackers all became the same. For brevity and with the exception of this protestation, we'll acquiesce to the same usage. There's an excellent discussion of the meaning of *hack* at the Jargon File site (*http://www.tuxedo.org/~esr/jargon/html/The-Meaning-of-Hack.html*).

leaving their unauthorized handprints inside. Already, you have a scenario where your potential opponents are much more motivated than you are. Your best hope is to be more rational than they are and expend your effort in the best possible way as you shore up your system security

Perhaps the only thing worse than not knowing and not fixing the vulnerabilities in your system is knowing and not fixing. It's simply not enough to know that you have some vulnerabilities in your system and that someday you'll get around to fixing them. We're proud of our systems and want to add features to move them forward. And we know that system security is one of those jobs that are invisible when done well. It very often boils down to working on security rather than adding new wonderful features to our systems. Someday, you'll be glad you did—when one of your peers at another site has to spend a week rebuilding everything from scratch because he didn't. Don't just learn about the flaws in your system—stay ahead of the curve by updating your software regularly. Apply the necessary patches, workarounds, and updates as necessary to keep your system beyond the threshold of likely hacker attention span.

Security Resources

Online informational security resources could loosely be grouped into three categories: hacker resources, neutral resources, and security professional resources. We won't attempt to make the value judgments necessary to group each site mentioned in this chapter into the categories. We'll just say that knowing that the different categories are there might help you decide with how much gravity to take information that you find at one of the sites.

Hacker resources often present themselves as free "services," while, in actuality, they post code that many twelve-year-olds could use to take down your site. Some of the "old guard" professional sites are somewhat hypersensitive where safety is concerned. They'll only post a warning about a vulnerability after each major vendor has had an opportunity to run their "spin" through the local legal and marketing departments, making the warning only marginally helpful once the information finally gets to you.

The Computer Emergency Response Team/ Coordination Center (CERT/CC)

http://www.cert.org/

Since 1988, from its home at the Carnegie Mellon Software Engineering Institute, CERT has stood watch over security on the Internet. Although CERT is regarded as somewhat slow on the draw these days in getting the word out, they performed

the difficult job of centralized security coordination back when no one else was there to do so. CERT® was formed as a response to the vulnerability pointed out by the Morris Worm attack (see *http://www.worm.net/*) that marked the first time many people ever heard of the Internet. CERT issues regular alerts in the form of advisories, summaries, and vendor-initiated bulletins, all of which are propagated through their *cert-advisory@cert.org* mailing list.

Advisories are time-sensitive notifications of events or recently discovered vulnerabilities that merit broad attention. *Summaries* are in-depth analyses of current threats on the Internet and advice on how best to address the threats. *Vendor-initiated* bulletins are postings provided by vendors who wish to send out security bulletins regarding their own products.

L0pht Heavy Industries

http://www.l0pht.com/

L0pht Heavy Industries is basically a bunch of hackers who put together a very polished full-disclosure web site that disseminates information about vulnerabilities—what systems have them, as well as how to exploit the vulnerabilities yourself. What this kind of site does is like practicing nuclear deterrence by giving everyone the missile codes and keys to the silos.

Fortunately or unfortunately, depending on your point of view, sites like this are the best source of information about newly discovered vulnerabilities. Usually, the first to discover an vulnerability will be hackers who may want to keep the vulnerability to themselves to give themselves a competitive edge over their peers. Before long, however, information about the vulnerabilities percolates out to full-disclosure sites like L0pht, RootShell, and Bugtraq, and sometime later, it eventually finds its way out to limited disclosure, industry-friendlier sites like CERT and CIAC.

Computer Incident Advisory Capability

http://ciac.llnl.gov/

The Computer Incident Advisory Capability (CIAC) is the U.S. Department of Energy's security tiger team. Their primary mission is to secure DOE facilities. Nevertheless, so much of what they do and the information they gather is of use to the general Internet community that their site has become a popular resource on the Internet over the years. One of the original security teams formed in the post–Morris Worm era, CIAC is useful to non-DOE folks mostly as a clearinghouse for warnings posted from other sites. Typically, CIAC will post CERT or vendor warnings but preface them with a sensible executive summary that boils the entire issue and its perceived urgency down to one screenful of useful information.

A couple of the more interesting features of the CIAC site are their hoaxes list (*http://ciac.llnl.gov/ciac/CIACHoaxes.html*) and chain letter list (*http://ciac.llnl.gov/ciac/CIACHoaxes.html*). CIAC collects data and stories on many Internet hoaxes and chain letters and keeps them as caveats.

RootShell

http://www.rootshell.com/

RootShell is arguably the most informative, commonly known site about vulnerabilities and events on the net. This full-disclosure site has reportedly been assigned partial blame by the U.S. Department of Defense for substantial increases in hacking attempts directed against DOD resources.

RootShell is, for the most part, a very attractive and useful site. The frequency with which vulnerabilities and methods are posted will keep anyone with a morbid interest in who's gotten hacked lately coming back for more. On a more practical note, RootShell propagates more than just plain-vanilla information on IP buffer overflow attacks. There's plenty of information on the site to encourage sleeplessness among the ranks of JavaScript users, Novell IPX network managers, MS-Windows SMB/CIFS users, anyone using a certain kind of UPS, anyone using certain kinds of wireless networking, anyone using certain kinds of printers—you get the idea. RootShell is food for your to-do list.

Bugtraq and SecurityFocus.com

http://securityfocus.com/

If you have time to keep up to date with only one security web site, *http://securityfocus.com* would be a good one to pick. The crown jewel of the site is the Bugtraq mailing list archives. Bugtraq content is much like RootShell, in that they're both full-disclosure sites. Bugtraq has much more of a system administrator flavor than RootShell, which is aimed more at an audience of hackers. As an interesting exercise, search the Bugtraq archive for keywords related to the systems you run (e.g., Solaris, sendmail, IMAP) and see what comes up.

Security Focus also has one of the very best archives around of security-related software. You might even consider it the Tucows or Hotfiles* of security software, with generous examples of access control, auditing, authentication, cryptography, intrusion detection, network monitoring, policy enforcement, programming, recovery, replacement, sniffers, system security, and other utilities. During our review of

* *http://www.tucows.com/* and *http://www.hotfiles.com/*, a couple of our favorite software archives.

this site, we found a nifty MS-Windows port of the Unix standard snooping utility, *tcpdump,* which captures and dissects just about any packet on your network.

A Handful of Security Tips

As much as we'd like to be comprehensive here, we won't. We can't. Nobody can. The nature of security is that it's always incomplete. As elusive as it is, 100% uptime is much easier to achieve than 100% security. The goal of this chapter, however, is to point you in some directions that will help you get as far above 99% secure as possible.

Having a secure site has a lot to do with not making any stupid little mistakes. A friend of ours once made the mistake of running an FTP session from a remote shell account provider to a local corporate site to get his *.cshrc* file, while under contract to a national ISP. Just that single occasion of grabbing a file turned into a situation where one of my accounts was compromised, using my password, and used to run an eggdrop* server. Those five seconds of indiscretion cost many hours of work by several people who had to pull the machine out of production, re-install the OS and all software, and put it back into production.

Assume all your unencrypted keystrokes are already in the hands of hackers. If that makes you feel uncomfortable, it ought to. Strong encryption hasn't propagated to all common applications, let alone all *uncommon* applications, yet. The best you can do sometimes is assume that you will be periodically compromised and take effective, routine protective measures.

Tripwire

A good example of one of those measures would be to run Tripwire. Tripwire, a host filesystem monitoring utility originally developed at Purdue University, retains a database of checksums and other data about files, directories, and devices throughout your system and notifies you if any of those entities you're watching changes. Free, old versions of Tripwire are available from Purdue University at *ftp://coast.cs.purdue.edu/pub/tools/unix/Tripwire/*. For-pay, commercial, new versions are available from Tripwire Security, Inc. at *http://www.tripwiresecurity.com/*.

Tripwire depends on having a reliable read-only copy of its database, usually on a write-protected floppy or a CD-ROM. Once installed on a clean system that has never been on the Net, it can be used reliably to notify you of the kinds of things a hacker might do to conceal her tracks. Some of those things might be modifying

* Eggdrop is a daemon used to automate maintenance of an IRC channel. In this case, it was likely being used as a file server for bootleg software and cracking information. One site with a fair amount of Eggdrop information is *http://www.roon.org/eggdrop/*.

utmp or *wtmp* databases, modifying executables like *finger*, *who*, *w*, or *ps*, or doing things like making */dev/kmem** world-readable.

Social Engineering

Another vulnerability to watch out for is social engineering. A friend of ours, also a former employee of the security group at a large ISP, called his new replacement, who didn't recognize his voice, and claimed to be the Unix system administration group leader. With polite apologies, he explained that he'd lost the slip of paper on which he had his root passwords written and asked the new guy if he would be so kind as to give him the passwords over the telephone. Not only did the former employee get the complete list of root passwords, but he also proceeded to call one of the Unix system administrators and conferenced in the new guy again to confirm the spelling of one of the passwords. It was an interesting day. Production essentially ceased while everyone scrambled to change the root passwords on all the machines immediately.

There are ways to engineer your system so that it's less vulnerable to social engineering. For example, instead of having all system administrators share the account named *root*, assign each person a root-equivalent account that could easily be turned off if an event necessitated it, without affecting root access for other system administrators.

The Man-in-the-Middle

For years, a major vulnerability that was found in most computing environments was intrinsic to the local area network itself: Ethernet. Ethernet is a broadcast network, and every station on the network can potentially see the traffic to and from every other station. That leaves the enormous potential for intermediate stations to capture copies of the traffic between legitimate hosts and use the information for illegitimate aims. A "Man in the Middle" attack refers to exploitation of information available to any intermediate point between the source and the destination of an IP packet. A passive man-in-the-middle attack might be the collection of passwords from cleartext logins. An active man-in-the-middle attack might involve active substitution of SSL or SSH encryption keys or capturing one end of a TCP session.

In fact, most cable modem arrangements and even some DSL setups are susceptible to this kind of exploit. With high-bandwidth, always-on Internet access becoming more prevalent, it's not unusual for users to find themselves the target of hack

* On some systems, */dev/kmem* is a file providing direct access to the entirety of virtual system memory, in which you can read and write anything in process space.

attempts within minutes of the time they first come online through their new cable modem or DSL connection. Even if you manage to successfully firewall your home network, the traffic going between your home and all the places you go on the Net may be considered fair game for hackers. It's best to assume that anything you send unencrypted on the Internet can, and will, be read by someone.

As an illustration, one of the colleges at a university where we worked recently started offering an entry-level class to train fledgling Unix system administrators. As a class exercise, everyone ran packet-gathering software to collect passwords for other users whose traffic might be passing through the network where their computer lab resides. That pretty much neutralized what little security was provided by any cleartext password scheme.

There are a few things you can do to greatly reduce the number of cleartext password transmissions on your network. One way is probably already being done at your site: using switches. With an Ethernet switch, as opposed to a hub, any given port only sees unicast* traffic that is destined for devices on that port. The closer you can get to achieving a 1:1 ratio of devices to switch ports, the more resistant your system will be to packet-capturing from intermediate systems. This doesn't reduce the damage that could be done if someone manages to capture your traffic, but it does significantly reduce the likelihood that traffic will be caught.

Even on a switched network, though, malicious users on the source or destination machine can surreptitiously obtain privileges on the host, capture all the packets sent to that host, and use them for ill gain. Overall, it's better to have numerous, modestly appointed hosts serving a single purpose than a small number of superhosts serving multiple purposes each. That is true if for no other reason than that a smaller user population for each host represents a smaller number of sources for locally based attacks.

TCP Wrappers

One way to make a substantial contribution to the security of your IMAP server is to run the TCP Wrappers package.† We'll cover a few of the more prominent applications and features of TCP Wrappers here, but you'd be well advised to read the manpages in depth for a more complete understanding.

---

* For the uninitiated, network traffic can be divided into three types: unicast, broadcast, and multicast. Unicast traffic is between one single machine and another single machine. Broadcast traffic is sent from a single machine to a group of machines on a single local area network. Multicast traffic is sent from a single machine to a group of machines that may reside on two or more geographically dispersed local area networks.

† The TCP Wrappers package is available from *ftp://coast.cs.purdue.edu/pub/tools/unix/tcp_wrappers*.

The name TCP Wrappers is actually a misnomer. The TCP Wrappers package is more of a border guard than a wrapper. TCP Wrappers monitors incoming requests for TCP services, such as Telnet, FTP, POP, and IMAP, that have a one-to-one mapping onto executable files. When *inetd* receives a request for a service, it's tricked into running the TCP Wrappers program (*tcpd*) instead of the server program (for example, */usr/cyrus/bin/imapd*). The TCP Wrappers program logs the request and checks its configuration to see whether or not it is allowed to service the request. If everything's kosher, TCP Wrappers runs the server program and disposes of itself.

The distinction between a true wrapper and a TCP Wrappers–style "border guard" is important for a couple of reasons. First, TCP Wrappers can do nothing to protect your assets once it has decided to let a service daemon start. If an incoming request meets all the criteria set forth in *hosts.allow* and *hosts.deny*, then TCP Wrappers has served its entire purpose by checking the request and starting the service.

Second, TCP Wrappers is a poor tool to improve the security of daemons that may service requests from more than one remote process during their lifetime. Some HTTPDs, for example, are fired off once in *inetd*, service their first request, and then stick around to service additional requests.

Basic installation

Once you've pulled down, compiled, and done the first rough-cut of installing the TCP Wrappers package, the next step is to configure it. There's more than one way to implement TCP Wrappers. We'll just discuss the most robust and secure way. The first step is to modify your */etc/inetd.conf* file.

First, *inetd* needs to be tricked into running *tcpd* instead of a TCP service daemon. This means changing a line in */etc/inetd.conf*. If, for example, your *imapd* looks like the following line before TCP Wrappers is installed:

```
imap4  stream tcp nowait cyrus /usr/cyrus/bin/imapd imapd
```

then the TCP wrapped version would be something like this:

```
imap4  stream tcp nowait cyrus /usr/local/etc/tcpd /usr/cyrus/bin/imapd
```

In the modified version, incoming requests for the *imap4* service are handed off to */usr/local/etc/tcpd*. If *tcpd* decides that it's okay to service the *imap4* request, it will run */usr/cyrus/bin/imapd*.

Notice we've appended the full path to the process name of the *imapd* daemon in the modified version. This may not strictly be necessary, but we don't like to rely on the daemon we need being somewhere in the system path. If the default path

for root gets corrupted (e.g., someone adds "." to the path to make life easier for themselves), we want one fewer thing to chase down and fix.

Now that you've changed your *inetd.conf* file, do whatever is necessary on your system to put your new *inetd.conf* into production. On most Unix systems, that means sending a HUP signal to the *inetd* process.

Before you restart *inetd*, make sure that you don't already have files called */etc/hosts.allow* or */etc/hosts.deny* in place. If you do, make sure you know that they reflect the security policy you want to enact on that particular host. If you haven't tested the rules in those files, errors could result in essential services being denied to your users—not a good thing on a production system.

Once TCP Wrappers is in production and *tcpd* is being used to fire off your individual TCP service daemons, the lack of */etc/hosts.allow* and */etc/hosts.deny* should permit *all* traffic to continue through as if TCP Wrappers weren't installed. The only difference you should notice at this point is syslog messages from *tcpd* in the log files for various TCP processes indicating when connections have been established from remote hosts.

The access files: /etc/hosts.allow and /etc/hosts.deny

The access granted to remote processes on your machine by TCP Wrappers is controlled by two files: */etc/hosts.allow* and */etc/hosts.deny*. If they don't exist or are empty, no restrictions are applied to incoming connections. *hosts.allow* is examined before *hosts.deny*. The first time a rule matches the given circumstance of connection, *tcpd* acts and further examination is discontinued. This means that if you permit all hosts in *usnd.edu* in *hosts.allow* and deny *hoopal.usnd.edu* in *hosts.deny*, *tcpd* will never see the rule to deny *hoopal.usnd.edu*. As soon as the match is made for permitting *usnd.edu*, rule examination will cease.

This section only provides a thumbnail sketch of how to set up TCP Wrappers. The *hosts_access*(5) and *host_options*(5) manpages together provide documentation for the TCP Wrappers Host Access Control Language. We're only touching on the basic essentials here.

Both *hosts.allow* and *hosts.deny* have the same format. Each entry takes the form:

```
daemon_list : client_list : option : option ...
```

daemon_list is a list of services to which a particular rule applies. The list can consist of daemon process names listed as the right-most argument in *inetd.conf* lines (*in.telnetd*, *in.ftpd*, *imapd*) and of wildcards such as `ALL`.

client_list is a list of source names and/or addresses to which a particular rule applies. It can include hostnames, addresses, patterns, and wildcards. *vaxb.acs.unt.edu*, *.acs.unt.edu*, *.edu*, 10.0.1.2, 129.120.51.50/255.255.255.254, *root@ALL*, and

kwm@cray23.themullets.net are all examples of valid entries. Fully qualified domain names, portions thereof, IP addresses, IP subnets expressed as address with netmasks, and *users@hosts* are all valid. In that last example, the username would have to be verifiable via the IDENT protocol. Please bear in mind that IDENT is not a verifiably reliable method for determining the authentic identity of the person on the other end of a communications channel. A user could be running a hacked version of IDENT or could connect using a username that commonly exists on most systems (e.g., *root*).

option is a modifier to the rule such as *allow*, *deny*, *spawn*, *twist*, and *severity*. *allow* and *deny*, respectively, permit or prohibit the request from being serviced by your host. *twist* permits you to hand off sessions bound for a rule match to an alternate command. This would let you hand off internal requests for your company's website to an HTTP server that defaults to the intranet site and hand off external requests to your company's external HTTP server.

severity lets you change the syslog severity with which a given event/rule match is logged. This can be used as a key in */etc/syslog.conf* to direct given types of error messages to a file by themselves.

It's interesting to note that the keywords *allow* and *deny* permit the entire set of rules to be kept in one file, so you have a choice of two configuration file approaches. To keep things simple, we'll use one file, *hosts.deny*. Suppose you wanted to configure *hosts.deny* to perform the following work:

1. Deny all *telnet* connections from all sites, and display an informational banner message to anyone who tries to *telnet* into your machine.
2. Deny *imapd* connections from *hacker.someplace.org.*

You'd add these entries to your *hosts.deny*:

```
in.telnetd: ALL: banner /etc/banners : DENY
imapd: hacker.someplace.org
```

Granted, TCP Wrappers is a somewhat narrowly focused security package, but it does everything it attempts to do very well. There's not a lot of secondary documentation out there on TCP Wrappers. Fortunately, the manpages that come with the source distribution are well written and have a very good signal-to-noise ratio.

Our verdict: all IMAP servers (and arguably all Unix hosts in general) should use the TCP Wrappers package as one component of a broad security framework.

A Word Against Cleartext Passwords

Possibly the biggest security risk on your IMAP server is the constant, unrelenting transmission of many users' passwords across the net. When the user runs an

IMAP client on her local machine, the IMAP client sends the username and password in cleartext every time it contacts the server (assuming that the client and server don't support an encrypted authentication method). The username and password could travel a long way over the Internet between client and server, leaving many chances that they might be compromised somewhere along the way. Anything you can do to reduce this risk would be good. You can protect your passwords by tunneling IMAP through SSL or SSH or by using an encrypted authentication method, such as Kerberos or CRAM.

SSL

One way to reduce the risk, especially if you have a captive customer base as in a corporate intranet, is to distribute SSL-capable clients and permit *only* SSL access to your server. In Appendix B, *Adding SSL Support to IMAP*, we cover the procedure for adding SSL support to your IMAP server. Adding SSL to your IMAP server encrypts all passwords and content between the IMAP client and the server processes as well as insulates users from the details of encryption. All they need know is that by using certain IMAP clients, their email traffic is safe from useful capture on the network.

SSH

One option for secure communication between the IMAP client and server is to tunnel the communication inside the Secure Shell protocol. The difficulty in doing so is just enough that we don't recommend it for people who either aren't wizards or don't have access to wizards. The mechanics are easy enough, but it might be necessary to call on a wizard the first time a problem crops up.

Conceptually, it works like this. First, you install an SSH client on the local machine where you run your IMAP client. You use the SSH client to establish an SSH connection to the remote host where the IMAP server is running.* You also use the SSH client to establish a "listen" on a local port for IMAP requests. Here's the cool part: when you fire up your IMAP client, it connects to the IMAP port on *localhost*—your machine—instead of connecting to port 143 on a remote server machine.

The SSH client then forwards everything it receives on the local IMAP port through the SSH session, or tunnel, to the remote SSH daemon, which then forwards the data to the IMAP port on the remote host.

---

* The SSH tunnel only encrypts from the client to the remote host. If the IMAP server is not running on the remote host, then packets from the remote host to the IMAP server will not be encrypted.

How does the SSH daemon on the receiving end know what to do with all this IMAP information coming at it? Well, the information is part of the port-forwarding arrangement you gave the daemon when you first fired up the SSH session. For example, you'd invoke SSH from your client machine like this:*

```
client# ssh -f -L 143:localhost:143 kwm@serverhost tail -f /dev/null
```

The command must be invoked as *root* because root privilege is required to set up port forwarding. The *–f* option tells SSH to run in the background after port forwarding has been established. *–L localport:remotehost:remoteport* specifies that the given port on the local (client) host is to be forwarded to the given host and port on the remote side. In our example, we use port 143 on both the client and the host, but that's just for simplicity. In reality, you can use any port on the client that isn't already in use. The server port must be whichever port listens for IMAP requests (143 on most systems). Depending on the SSH client, you'll either be prompted for your password to log in to the server when issuing the tunneling command, or you'll have to initiate a login manually to establish the session. In all cases, you'll have to use SSH to log in to the remote host before you can use it to "launder" your connection. The entire IMAP port-forwarding scenario is shown in Figure 13-1.

If this all seems a little obtuse, don't worry. A couple of examples should help. We'll start with a command-line example. We start by using *lsof*† to check for software listening at local TCP port 143. There is none. We confirm this by trying to *telnet* to localhost at port 143 without success.

```
[kwm@clienthost]% lsof -i tcp:143              Lists all activity on port 143
[kwm@clienthost]% telnet localhost 143
Trying 127.0.0.1...
telnet: Unable to connect to remote host: Connection refused
```

At this point, we're certain that there's no activity, such as a listen or an open connection, on port 143 on our local machine. That port is okay to use. Next, we set up the port forwarding by issuing an SSH command. Remember that you have to be *root* to set up port forwarding:

```
[kwm@clienthost] % su -
Password:
[root@clienthost]# ssh -f -L 143:localhost:143 kwm@serverhost tail -f /dev/null
kwm@serverhost's password:
[root@clienthost]# ^D
```

* If you use a Windows or Mac SSH client such as TeraTerm, port forwarding is done through the windows interface.

† *lsof* (ls Open Files), a program that tells you which open files and network connections belong to which processes, is available at *ftp://vic.cc.purdue.edu/pub/tools/unix/lsof/*.

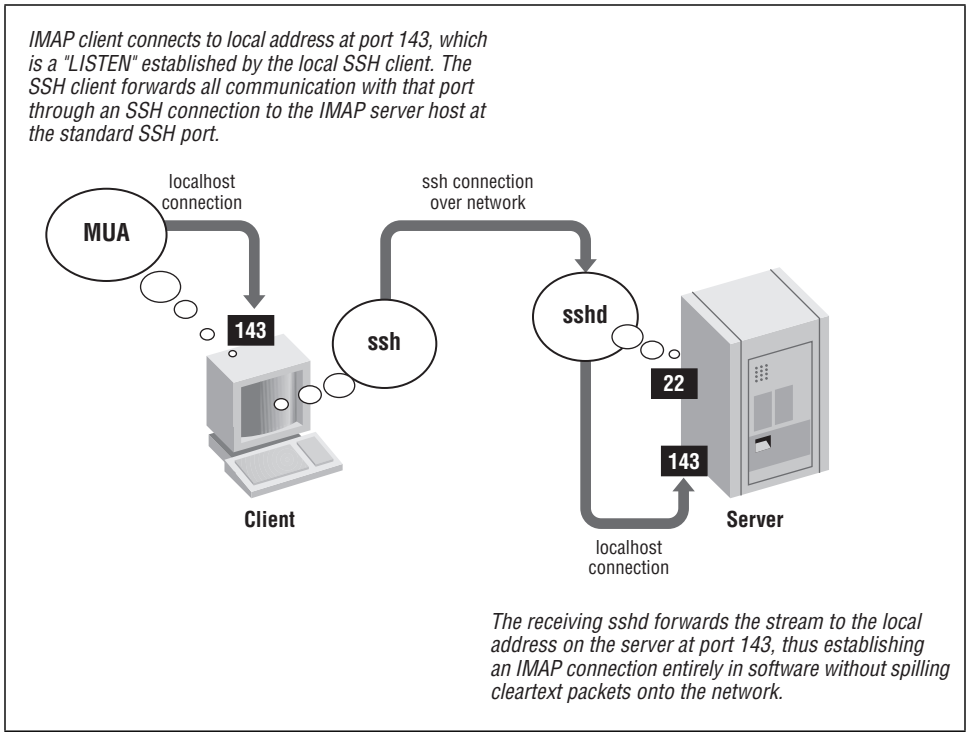

Figure 13-1. IMAP port forwarding through SSH

The *tail –f /dev/null* that we tacked on to the end of the SSH command is just a low-overhead command to keep the session open. We didn't want to keep an actual shell session open and running in the background when we didn't need it, so we used the *tail* command instead.

Port forwarding is active. Now, when we look at port 143 on our local machine, it's ready to accept IMAP connections:

```
[kwm@clienthost]% lsof -i tcp:143
COMMAND  PID USER   FD   TYPE DEVICE SIZE NODE NAME
ssh     1958 root    4u  IPv4  45438       TCP localhost:imap (LISTEN)
[kwm@clienthost]% telnet localhost 143
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
* OK localhost IMAP4rev1 v12.250 server ready
. capability
* CAPABILITY IMAP4 IMAP4REV1 NAMESPACE IDLE SCAN SORT MAILBOX-REFERRALS
LOGIN-REFERRALS AUTH=LOGIN AUTH=ANONYMOUS THREAD=ORDEREDSUBJECT
. OK Completed
. logout
* BYE serverhost IMAP4rev1 server terminating connection
. OK Completed
Connection closed by foreign host.
```

Although we're connected to the IMAP server port on *clienthost*, the goodbye message from the IMAP server identifies the server as *serverhost*. Users can now point their IMAP clients at port 143 on *clienthost* and have their IMAP sessions encrypted between *clienthost* and *serverhost*. If their IMAP clients are actually running on *clienthost*, their IMAP traffic is encrypted. This kind of arrangement may also be useful for remote campuses needing to participate in an enterprise's intranet by tunneling sensitive information (email over IMAP, in this case) within encrypted tunnels (SSH, here).

Encrypted authentication: Kerberos and CRAM

Another good way to provide password security is though an encryption method such as CRAM or Kerberos. CRAM encrypts the password only, and Kerberos relies on the encryption of Kerberos tickets so, strictly speaking, your password isn't going out over the Net. CRAM is less desirable than SSL because the transmission of an obviously encrypted password string amounts to hacker bait for someone with the tools to attempt cracking the encryption. Kerberos is less desirable than SSL simply because it has the added requirement of bringing up a Kerberos server.

The Core of the Problem

Some implementations of IMAP retain security information, such as passwords, in the memory space of the *imapd* process. For that reason, it's a good idea to set your maximum corefile size to zero for the *imapd* process. Because *imapd* is usually run out of *inetd*, this means setting the limit in the startup script that starts *inetd*. On a Solaris system, for example, the startup script would be found under */etc/init.d* and would be a Bourne shell script. In the Bourne shell, this is done with the command:

```
ulimit -c 0
```

Once you've done that, malicious folks can't covertly trigger a coredump in an *imapd* process and salvage password information from its contents.

Monitoring Security

We recommend a two-pronged strategy for the monitoring necessary to keep apprised of the quality of security on your network and IMAP server. First, run a variety of tools that let you observe, at a low level, the character of the traffic on your network. Second, never be in doubt about the status of your services. Know beyond any doubt if every server of yours is up or down and, by extension, if every service is up or down. Hopefully, you have the resources to engineer your Internet services so that the failure of one or two servers doesn't negatively impact the status of the service they provide.

IP Watcher

http://www.engarde.com/software/ipwatcher/

IP Watcher is a slick application that displays, in either an X or a Curses application, a list of all the current TCP-based sessions in progress and permits you to observe or disconnect them. IP Watcher is a tool you may never need. If, however, you have an incursion on your network and the hacker's already on the premises, there's a chance you might be able to gather more evidence if you have IP Watcher handy.

NetLog

http://www.net.tamu.edu/ftp/security/TAMU/netlog.README

NetLog is at the other end of the spectrum from IP Watcher. While IP Watcher excels at watching what a single person is doing on the network, right down to duplicating the contents of her Telnet session screen, NetLog lets you characterize the usage of your network over time. Think of IP Watcher as a single phone tap and NetLog as the National Security Agency.

NetLog consists of four packages, *tcplogger*, *udplogger*, *extract*, and *NetWatch*. *tcplogger* and *udplogger* log TCP and UDP sessions on the locally visible network. *extract* pulls information out of the logs they produce, and *NetWatch* is a real-time monitor: a more statistically oriented version of IP Watcher.

Using the information from *tcplogger* and a log-triggering package like *swatch* (see the next section), you can send yourself a page if a port scan starts or other weird traffic starts developing against your hosts.

swatch

ftp://coast.cs.purdue.edu/pub/tools/unix/swatch/

swatch is one of those packages that ought to be on every Unix box. If you've administered a host for any length of time, you've been there. The log files, extracts from log files, and renamed log files are slowly taking over your disk space. They're full of good information about system events, but more often than not, you find out on Monday that something went wrong on Friday at 5:30. *swatch* observes your log files, watches for regular expressions you define, and notifies you in one of about a zillion ways without getting carried away. If a given event causes 50 log messages, you can choose to be notified only once. Not surprisingly, *swatch* is written in Perl.

Network Operations Center On-Line (NOCOL)

http://www.netplex-tech.com/software/nocol/

If you ply your trade at a site of any decent size, you probably have a fair amount of resources dedicated to high-end network management packages like HP Openview, SunNet Manager, and Cabletron Spectrum. All those things are fine and dandy for configuring or pulling statistics from your various devices. If what you really want, though, is a status screen for you and your operators to tell at a glance what's running and what's not, NOCOL is very likely just the ticket.

NOCOL consists of command-line, Curses, Web, and multiple API interfaces into a single state engine. That state engine comes with monitors for ICMP ping, RPC portmapper, OSI ping, Ethernet load, TCP ports, nameserver, radius server, syslog messages, mail queue, NTP, UPS (APC) battery, Unix host performance, BGP peers, SNMP variables, and overall host data throughput. Additional probes are easily written. Rather than be in binary UP or DOWN states, each monitored service can be in info (up), warning, error, or critical states that are definable for each service.

Boiling It All Down

Essentially, in order to care for the security of your hosts and the services, you have to be both a psychologist and a sociologist. That applies to your hosts, their users, and the abusers as well. Knowing how all the processes and resources within a host interact is as important as knowing how the host interacts with other hosts on the network. Anticipating what your users are going to do next has to be balanced with time spent trying to second-guess the next hacker with too much spare time.

You're probably going to spend either too much or too little time on security. There's no way of knowing ahead of time if the risks merit the effort—that's the nature of insurance policies.