6 Security

In a traditional wired network, access control is very straightforward: If a person has physical access to a computer or network hub, then they can use (or abuse) the network resources. While software mechanisms are an important component of network security, limiting physical access to the network devices is the ultimate access control mechanism. Simply put, if all terminals and network components are only accessible to trusted individuals, then the network can likely be trusted.

The rules change significantly with wireless networks. While the apparent range of your access point may seem to be just a few hundred meters, a user with a high gain antenna may be able to make use of the network from several blocks away. Should an unauthorized user be detected, is impossible to simply “trace the cable” back to the user’s location. Without transmitting a single packet, a nefarious user can even log all network data to disk. This data can later be used to launch a more sophisticated attack against the network. Never assume that radio waves simply “stop” at the edge of your property line.

Of course, even in wired networks, it’s never quite possible to completely trust all users of the network. Disgruntled employees, uneducated network users, and simple mistakes on the part of honest users can cause significant harm to network operations. As the network architect, your goal should be to facilitate private communication between legitimate users of the network. While a certain amount of access control and authentication is necessary in any network, you have failed in your job if legitimate users find it difficult to use the network to communicate.

There’s an old saying that the only way to completely secure a computer is to unplug it, lock it in a safe, destroy the key, and bury the whole thing in concrete. While such a system might be completely “secure”, it is useless for communication. When you make security decisions for your network, remember that above all else, the network exists so that its users can communicate with each other. Security considerations are important, but should not get in the way of the network's users.

Physical security

When installing a network, you are building an infrastructure that people will depend on. And thus, the network must be reliable. For many installations, outages often occur due to human tampering, accidental or not. Networks are physical, wires and boxes, things that are easily disturbed. In many installations, people will not know what the equipment is that you have installed, or, curiosity leads them to experiment. They will not realize the importance that a cable goes to a port. People might move an Ethernet cable so that they can connect their laptop for 5 minutes, or move a switch because it is in their way. A plug might be removed from a power bar because someone needs that receptacle. Assuring the physical security of an installation is paramount. Signs and labels will only be useful to few, whom can read, or speak your language. Putting things out of the way, and limiting access is the best means to assure that accidents, or tinkering does not occur.

In less developed economies proper fasteners, ties, or boxes will not be as easy to find. You should be able to find electrical supplies that will work just as well. Custom enclosures are also easy to manufacture and should be considered essential to any installation. It is often economical to pay a mason to make holes and install conduit, where this would be an expensive option in the developed world, this type of labour intensive activity can be affordable in Southern countries. PVC can be embedded in cement walls for passing cable from room to room, this avoids smashing holes every time a cable needs to be passed. To insulate, plastic bags can be stuffed into the conduit around the cables.

Small equipment should be mounted on the wall and larger equipment should be put in a closet or in a cabinet.

Switches

Switches, hubs or interior access points can with a wall plug be screwed directly onto a wall. Best to put this equipment as high as possible to reduce the chance that someone will touch the device or its cables.

Cables

Cables should be hidden and fastened. Better to bury cables, than to leave them hanging across a yard, where it might be used for drying clothes, or simply snagged by a ladder etc. To avoid vermin and insects find plastic electrical conduit. The marginal expense will be well worth the trouble. The conduit should be buried about 30cm deep (below the frost in cold climates). It is also worth buying larger conduit than is presently required, so that future cables can be run through the same tubing. It is also possible to find plastic cable conduit that can be used in buildings. If not, simple cable attachments, nailed into the wall can be used to secure the cable and to make sure that it doesn't hang where it can be snagged, pinched or cut.

Power

It is best to have power bars locked in a cabinet. If that is not possible, Mount the power bar under a desk, or on the wall and use duct tape (gaffer tape, a strong adhesive tape) to secure the plug into the receptacle. On the UPS and power bar, do not leave empty receptacles, tape them if necessary. People will have the tendency to use the easiest receptacle, so make these critical ones difficult to use. If you do not, you might find a fan or light plugged into your UPS; though it is nice to have light, it is nicer to keep your server running!

Water

Protect your equipment from water and moisture. In all cases make sure that your equipment, including your UPS is at least 30cm from the ground, to avoid flooding. Also try to have a roof over your equipment, so that water and moisture will not fall onto it. In moist climates, it is important that the equipment has proper ventilation to assure that moisture can be exhausted. Small closets need to have ventilation, or moisture and heat can degrade or destroy your gear.

Masts

Equipment installed on a mast is often safe from thieves. Nonetheless, to deter thieves and to keep your equipment safe from winds it is good to over-engineer mounts. Equipment should be painted a dull, white or grey colour to reflect the sun and to make it look plain and uninteresting. Panel antennas are much more subtle and less interesting than dishes and thus should be preferred. Any installation on walls, should require a ladder to reach. Try choosing well lit but not prominent places to put equipment. Also avoid antennae that resemble television antennae, as those are items that will attract interest by thieves, where a wifi antenna will be useless to the average thief.

Threats to the network

One critical difference between Ethernet and wireless is that wireless networks are built on a ***shared medium***. They more closely resemble the old network hubs than modern switches, in that every computer connected to the network can “see” the traffic of every other user. To monitor all network traffic on an access point, one can simply tune to the channel being used, put the network card into monitor mode, and log every frame. This data might be directly valuable to an eavesdropper (including data such as email, voice data, or online chat logs). It may also provide passwords and other sensitive data, making it possible to compromise the network even further. As we’ll see later in this chapter, this problem can be mitigated by the use of encryption.

Another serious problem with wireless networks is that its users are relatively ***anonymous***. While it is true that every wireless device includes a unique MAC address that is supplied by the manufacturer, these addresses can often be changed with software. Even given the MAC address, it can be very difficult to judge where a wireless user is physically located. Multipath effects, high gain antennas, and widely varying radio transmitter characteristics can make it impossible to determine if a malicious wireless user is sitting in the next room or is in an apartment building a mile away.

While unlicensed spectrum provides a huge cost savings to the user, it has the unfortunate side effect that ***denial of service*** (***DoS***) attacks are trivially simple. By simply turning on a high powered access point, cordless phone, video transmitter, or other 2.4GHz device, a malicious person could cause significant problems on the network. Many network devices are vulnerable to other forms of denial of service attacks as well, such as disassociation flooding and ARP table overflows.

Here are several categories of individuals who may cause problems on a wireless network:

- ***Unintentional users***. As more wireless networks are installed in densely populated areas, it is common for laptop users to accidentally associate to the wrong network. Most wireless clients will simply choose any available wireless network when their preferred network is unavailable. The user may then make use of this network as usual, completely unaware that they may be transmitting sensitive data on someone else’s network. Malicious people may even take advantage of this by setting up access points in strategic locations, to try to attract unwitting users and capture their data.

  The first step in avoiding this problem is educating your users, and stressing the importance of connecting only to known and trusted networks. Many wireless clients can be configured to only connect to trusted networks, or to ask permission before joining a new network. As we will see later in this chapter, users can safely connect to open public networks by using strong encryption.

- ***War drivers***. The "war driving" phenomenon draws its name from the popular 1983 hacker film, "War Games". War drivers are interested in finding the physical location of wireless networks. They typically drive around with a laptop, GPS, and omnidirectional antenna, logging the name and location of any networks they find. These logs are then combined with logs from other war drivers, and are turned into graphical maps depicting the wireless "footprint" of a particular city.

  The vast majority of war drivers likely pose no direct threat to networks, but the data they collect might be of interest to a network cracker. For example, it might be obvious that an unprotected access point detected by a war driver is located inside a sensitive building, such as a government or corporate office. A malicious person could use this information to illegally access the network there. Arguably, such an AP should never have been set up in the first place, but war driving makes the problem all the more urgent. As we will see later in this chapter, war drivers who use the popular program NetStumbler can be detected with programs such as Kismet. For more information about war driving, see sites such as *http://www.wifimaps.com/*, *http://www.nodedb.com/*, or *http://www.netstumbler.com/* .

- ***Rogue access points***. There are two general classes of rogue access points: those incorrectly installed by legitimate users, and those installed by malicious people who intend to collect data or do harm to the network. In the simplest case, a legitimate network user may want better wireless coverage in their office, or they might find security restrictions on the corporate wireless network too difficult to comply with. By installing an inexpensive consumer access point without permission, the user opens the entire network up to potential attacks from the inside. While it is possible to scan for unauthorized access points on your wired network, setting a clear policy that prohibits them is very important.

  The second class of rogue access point can be very difficult to deal with. By installing a high powered AP that uses the same ESSID as an existing network, a malicious person can trick people into using their equipment, and log or even manipulate all data that passes through it. Again, if your users are trained to use strong encryption, this problem is significantly reduced.

- ***Eavesdroppers***. As mentioned earlier, eavesdropping is a very difficult problem to deal with on wireless networks. By using a passive monitoring tool (such as Kismet), an eavesdropper can log all network data from a great distance away, without ever making their presence known. Poorly encrypted data can simply be logged and cracked later, while unencrypted data can be easily read in real time.

If you have difficulty convincing others of this problem, you might want to demonstrate tools such as Etherpeg (*http://www.etherpeg.org/*) or Driftnet (*http://www.ex-parrot.com/~chris/driftnet/*). These tools watch a wireless network for graphical data, such as GIF and JPEG files. While other users are browsing the Internet, these tools simply display all graphics found in a graphical collage. I often use tools such as this as a demonstration when lecturing on wireless security. While you can tell a user that their email is vulnerable without encryption, nothing drives the message home like showing them the pictures they are looking at in their web browser.

Again, while it cannot be completely prevented, proper application of strong encryption will discourage eavesdropping.

This introduction is intended to give you an idea of the problems you are up against when designing a wireless network. Later in this chapter, we will look at tools and techniques that will help you to mitigate these problems.

Authentication

Before being granted access to network resources, users should first be ***authenticated***. In an ideal world, every wireless user would have an identifier that is unique, unchangeable, and cannot be impersonated by other users. This turns out to be a very difficult problem to solve in the real world.

The closest feature we have to a unique identifier is the MAC address. This is the 48-bit number assigned by the manufacturer to every wireless and Ethernet device. By employing ***mac filtering*** on our access points, we can authenticate users based on their MAC address. With this feature, the access point keeps an internal table of approved MAC addresses. When a wireless user tries to associate to the access point, the MAC address of the client must be on the approved list, or the association will be denied. Alternately, the AP may keep a table of known “bad” MAC addresses, and permit all devices that are not on the list.

Unfortunately, this is not an ideal security mechanism. Maintaining MAC tables on every device can be cumbersome, requiring all client devices to have their MAC addresses recorded and uploaded to the APs. Even worse, MAC addresses can often be changed in software. By observing MAC addresses in use on a wireless network, a determined attacker can “spoof” an approved MAC address and successfully associate to the AP. While MAC filtering will prevent unintentional users and even most curious individuals from accessing the network, MAC filtering alone cannot prevent attacks from determined attackers.

MAC filters are useful for temporarily limiting access from misbehaving clients. For example, if a laptop has a virus that sends large amounts of spam or other traffic, its MAC address can be added to the filter table to stop the traffic immediately. This will buy you time to track down the user and fix the problem.

Another popular authentication feature of wireless the so-called ***closed network***. In a typical network, APs will broadcast their ESSID many times per second, allowing wireless clients (as well as tools such as NetStumbler) to find the network and display its presence to the user. In a closed network, the AP does not beacon the ESSID, and users must know the full name of the network before the AP will allow association. This prevents casual users from discovering the network and selecting it in their wireless client.

There are a number of drawbacks to this feature. Forcing users to type in the full ESSID before connecting to the network is error prone and often leads to support calls and complaints. Since the network isn't obviously present in site survey tools like NetStumbler, this can prevent your networks from showing up on war driving maps. But it also means that other network builders cannot easily find your network either, and specifically won't know that you are already using a given channel. A conscientious neighbor may perform a site survey, see no nearby networks, and install their own network on the same channel you are using. This will cause interference problems for both you and your neighbor.

Finally, using closed networks ultimately adds little to your overall networks security. By using passive monitoring tools (such as Kismet), a skilled user can detect frames sent from your legitimate clients to the AP. These frames necessarily contain the network name. A malicious user can then use this name to associate to the access point, just like a normal user would.

Encryption is probably the best tool we have for authenticating wireless users. Through strong encryption, we can uniquely identify a user in a manner that is very difficult to spoof, and use that identity to determine further network access. Encryption also has the benefit of adding a layer of privacy by preventing eavesdroppers from easily watching network traffic.

The most widely employed encryption method on wireless networks is ***WEP encryption***. WEP stands for ***wired equivalent privacy***, and is supported by virtually all 802.11a/b/g equipment. WEP uses a shared 40-bit key to encrypt data between the access point and client. The key must be entered on the APs as well as on each of the clients. With WEP enabled, wireless clients cannot associate with the AP until they use the correct key. An eavesdropper listening to a WEP-enabled network will still see traffic and MAC addresses, but the data payload of each packet is encrypted. This provides a fairly good authentication mechanism while also adding a bit of privacy to the network.

WEP is definitely not the strongest encryption solution available. For one thing, the WEP key is shared between all users. If the key is compromised (say, if one user tells a friend what the password is, or an employee is let go) then changing the password can be prohibitively difficult, since all APs and client devices need to be changed. This also means that legitimate users of the network can still eavesdrop on each others' traffic, since they all know the shared key.

The key itself is often poorly chosen, making offline cracking attempts feasible. Even worse, the implementation of WEP itself is broken in many implementations, making it even easier to crack some networks. While manufacturers have implemented a number of extensions to WEP (such as longer keys and fast rotation schemes), these extensions are not part of the standard, and will not interoperate between equipment from different manufacturers. By upgrading to the most recent firmware for all of your wireless devices, you can prevent some of the early attacks found in WEP.

WEP can still be a useful authentication tool. Assuming your users can be trusted not to give away the password, you can be fairly sure that your wireless clients are legitimate. While WEP cracking is possible, it is beyond the skill of most users. WEP is extremely useful for securing long distance point-to-point links, even on generally open networks. By using WEP on such a link, you will discourage others from associating to the link, and they will likely use other available APs instead. WEP is definitely a handy "keep out" sign for your network. Anyone who detects the network will see that a key is required, making it clear that they are not welcome to use it.

WEPs greatest strength is its interoperability. In order to comply with the standards, all wireless devices support basic WEP. While it isn't the strongest method available, it is certainly the most commonly implemented feature. We will look at other more advanced encryption techniques later in this chapter.

For more details about the state of WEP encryption, see these papers:

- http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html
- http://www.cs.umd.edu/~waa/wireless.pdf
- http://www.crypto.com/papers/others/rc4_ksaproc.ps

Another data-link layer authentication protocol is ***Wi-Fi Protected Access***, or ***WPA***. WPA was created specifically to deal with the known problems with WEP mentioned earlier. It provides a significantly stronger encryption scheme, and can use a shared private key, unique keys assigned to each user, or even SSL certificates to authenticate both the client and the access point. Authentication credentials are checked using the 802.1X protocol, which can consult a third party database such as RADIUS. Through the use of Temporal Key Integrity Protocol (TKIP), keys can be rotated quickly over time, further reducing the likelihood that a particular session can be cracked. Overall, WPA provides significantly better authentication and privacy than standard WEP.

The difficulty with WPA is that, as of this writing, interoperability between vendors is still very low. WPA requires fairly recent access point hardware and up-to-date firmware on all wireless clients, as well as a substantial amount of configuration. If you are installing a network in a setting where you control the entire hardware platform, WPA can be ideal. By authenticating both clients and APs, it solves the rogue access point problem and provides many significant advantages over WEP. But in most network settings where the vintage of hardware is mixed and the knowledge of wireless users is limited, WPA can be a nightmare to install. It is for this reason that most sites continue to use WEP, if encryption is used at all.

Captive portals

One common authentication tool used on wireless networks is the ***captive portal***. A captive portal uses a standard web browser to give a wireless user the opportunity to present login credentials. It can also be used to present information (such as an Acceptable Use Policy) to the user before granting further access. By using a web browser instead of a custom program for authentication, captive portals work with virtually all laptops and operating systems. Captive portals are typically used on open networks with no other authentication methods (such as WEP or MAC filters).

To begin, a wireless user opens their laptop and selects the network. Their computer requests a DHCP lease, which is granted. They then use their web browser to go to any site on the Internet.

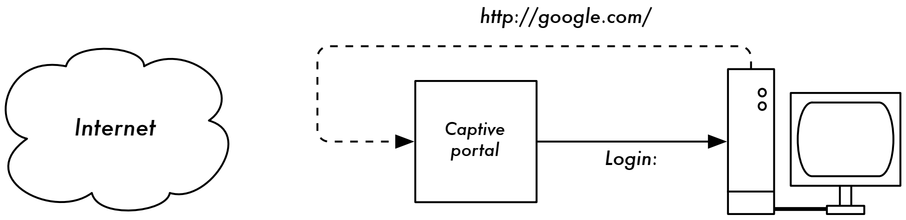

Figure 6.1: The user requests a web page and is redirected.

Instead of receiving the requested page, the user is presented with a login screen. This page can require the user to enter a user name and password, simply click a “login” button, type in numbers from a pre-paid ticket, or enter any other credentials that the network administrators require. The user then enters their credentials, which are checked by the access point or another server on the network. All other network access is blocked until these credentials are verified.

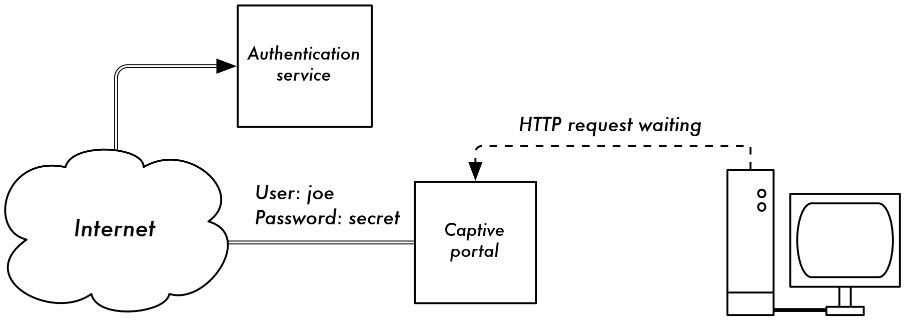

Figure 6.2: The user's credentials are verified before further network access is granted. The authentication server can be the access point itself, another machine on the local network, or a server anywhere on the Internet.

Once authenticated, the user is permitted to access network resources, and is typically redirected to the site they originally requested.

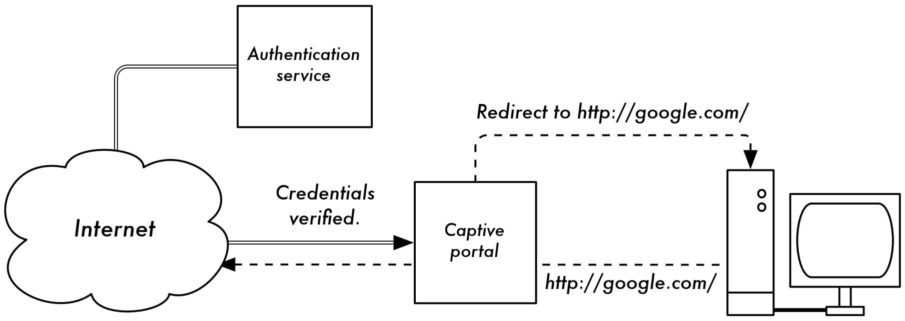

Figure 6.3: After authenticating, the user is permitted to access the rest of the network.

Captive portals provide no encryption for the wireless users, instead relying on the MAC and IP address of the client as a unique identifier. Since this is not necessarily very secure, many implementations will require the user to re-authenticate periodically. This can often be automatically done by minimizing a special pop-up browser window when the user first logs in.

Since they do not provide strong encryption, captive portals are not a very good choice for networks that need to be locked down to only allow access from trusted users. They are much more suited to cafes, hotels, and other public access locations where casual network users are expected.

In public or semi-public network settings, encryption techniques such as WEP and WPA are effectively useless. There is simply no way to distribute public or shared keys to members of the general public without compromising the security of those keys. In these settings, a simple application such as a captive portal provides a level of service somewhere between completely open and completely closed.

Two popular open source captive portal implementations are NoCatSplash and Chillispot.

NoCatSplash

If you need to simply provide users of an open network with information and an acceptable use policy, take a look at NoCatSplash. It is available online at *http://nocat.net/download/NoCatSplash/* .

NoCatSplash provides a customizable splash page to your users, requiring them to click a “login” button before using the network. This is useful for identifying the operators of the network and displaying rules for network access.

NoCatSplash is written in C, and will run on just about any Unix-like operating system including Linux, BSD, and even embedded platforms such as OpenWRT. It has a simple configuration file and can serve any custom HTML file as the splash page. It is typically run directly on an access point, but can also work on a router or proxy server. For more information, see *http://nocat.net/* .

Other popular hotspot projects

NoCatSplash is just one simple captive portal implementation. Many other free implementations exist that support a diverse range of functionality. Some of these include:

- Chillispot (*http://www.chillispot.org/*). Chillispot is a captive portal designed to authenticate against an existing user credentials database, such as RADUIS. Combined with the application phpMyPrePaid, pre-paid ticket based authentication can be implemented very easily You can download phpMyPrePaid from *http://sourceforge.net/projects/phpmyprepaid/*.
- WiFi Dog (*http://www.wifidog.org/*). WiFi Dog provides a very complete captive portal authentication package in very little space (typically under 30kb). From a user’s perspective, it requires no pop-up or javascript support, allowing it to work on a wider variety of wireless devices.
- m0n0wall (*http://m0n0.ch/wall/*). As mentioned in chapter five, m0n0wall is a complete embedded operating system based on FreeBSD. It includes a captive portal with RADIUS support, as well as a PHP web server.

Privacy

Most users are blissfully unaware that their private email, chat conversations, and even passwords are often sent "in the clear" over dozens of untrusted networks before arriving at their ultimate destination on the Internet. However mistaken they may be, users still typically have some expectation of privacy when using computer networks.

Privacy can be achieved, even on untrusted networks such as public access points and the Internet. The only proven effective method for protecting privacy is the use of strong ***end-to-end encryption***.

Encryption techniques such as WEP and WPA attempt to address the privacy issue at layer two, the data-link layer. While this does protect eavesdroppers from listening in on the wireless connection, protection ends at the access point. If the wireless client uses insecure protocols (such as POP or simple SMTP for receiving and sending email), then users beyond the AP can still log the session and see the sensitive data. As mentioned earlier, WEP also suffers from the fact that it uses a shared private key. This means that legitimate wireless users can eavesdrop on each other, since they all know the private key.

By using encryption to the remote end of the connection, users can neatly sidestep the entire problem. These techniques work well even on untrusted public networks, where eavesdroppers are listening and possibly even manipulating data coming from the access point.

To ensure data privacy, good end-to-end encryption should provide the following features:

- ***Verified authentication of the remote end***. The user should be able to know without a doubt that the remote end is who it claims to be. Without authentication, a user could give sensitive data to anyone claiming to be the legitimate service.
- ***Strong encryption methods***. The encryption algorithm should stand up to public scrutiny, and it should not be easily decrypted by a third party. There is no security in obscurity, and strong encryption is even stronger when the algorithm is widely known and subject to peer review. A good algorithm with a suitably large and protected key can provide encryption that is unlikely to be broken by any effort in our lifetimes using current technology.
- ***Public key cryptography***. While not an absolute requirement for end-to-end encryption, the use of public key cryptography instead of a shared key can ensure that an individual user's data remains private, even if the key of

another user of the service is compromised. It also solves certain problems with distributing keys to users over untrusted networks.

- ***Data encapsulation***. A good end-to-end encryption mechanism protects as much data as possible. This can range from encrypting a single email transaction to encapsulation of all IP traffic, including DNS lookups and other supporting protocols. Some encryption tools simply provide a secure channel that other applications can use. This allows users to run any program they like and still have the protection of strong encryption, even if the programs themselves don't support it.

Be aware that laws regarding the use of encryption vary widely from place to place. Some countries treat encryption as munitions, and may require a permit, escrow of private keys, or even prohibit its use altogether. Before implementing any solution that involves encryption, be sure to verify that use of this technology is permitted in your local area.

In the following sections, we'll take a look at some specific tools that can provide good protection for your users' data.

SSL

The most widely available end-to-end encryption technology is ***Secure Sockets Layer***, known simply as ***SSL***. Built into virtually all web browsers, SSL uses public key cryptography and a trusted ***public key infrastructure*** (***PKI***) to secure data communications on the web. Whenever you visit a web URL that starts with https, you are using SSL.

The SSL implementation built into web browsers includes a collection of certificates from trusted sources, called ***certificate authorities*** (***CA***). These certificates are cryptographic keys that are used to verify the authenticity of websites. When you browse to a website that uses SSL, the browser and the server first exchange certificates. The browser then verifies that the certificate provided by the server matches its DNS host name, that it has not expired, and that it is signed by a trusted certificate authority. The server optionally verifies the identity of the browser's certificate. If the certificates are approved, the browser and server then negotiate a master session key using the previously exchanged certificates to protect it. That key is then used to encrypt all communications until the browser disconnects. This kind of data encapsulation is known as a ***tunnel***.

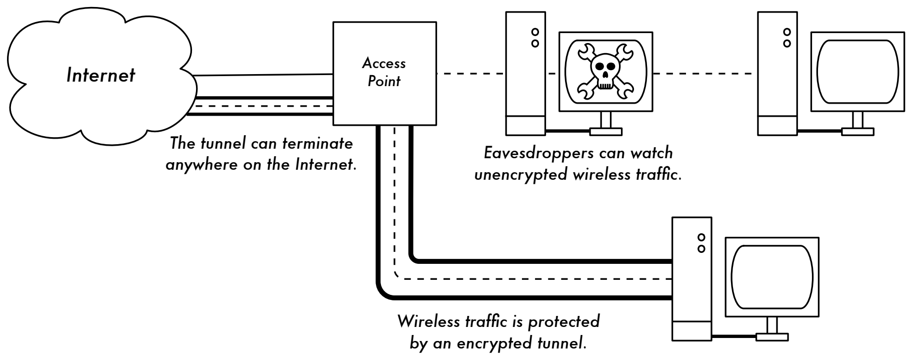
Internet
Access Point
The tunnel can terminate anywhere on the Internet.
Eavesdroppers can watch unencrypted wireless traffic.
Wireless traffic is protected by an encrypted tunnel.

Figure 6.4: Eavesdroppers must break strong encryption to monitor traffic over an encrypted tunnel. The conversation inside the tunnel is identical to any other unencrypted conversation.

The use of certificates with a PKI not only protects the communication from eavesdroppers, but prevents so-called ***man-in-the-middle*** (***MITM***) attacks. In a man-in-the-middle attack, a malicious user intercepts all communication between the browser and the server. By presenting bogus certificates to both the browser and the server, the malicious user could carry on two simultaneous encrypted sessions. Since the malicious user knows the secret on both connections, it is trivial to observe and manipulate data passing between the server and the browser.

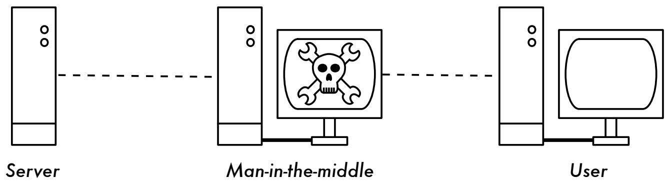

Figure 6.5: The man-in-the-middle effectively controls everything the user sees, and can record and manipulate all traffic. Without a public key infrastructure to verify the authenticity of keys, strong encryption alone cannot protect against this kind of attack.

Use of a good PKI prevents this kind of attack. In order to be successful, the malicious user would have to present a certificate to the client that is signed by a trusted certificate authority. Unless a CA has been compromised (very unlikely) or the user can be tricked into accepting the bogus certificate, then such an attack is not possible. This is why it is vitally important that users understand that ignoring warnings about expired or bogus certificates is very dangerous, especially when using wireless networks. By clicking the "ignore" button when prompted by their browser, users open themselves up to many potential attacks.

SSL is not only used for web browsing. Insecure email protocols such as IMAP, POP, and SMTP can be secured by wrapping them in an SSL tunnel. Most modern email clients support IMAPS and POPS (secure IMAP and POP) as well as SSL/TLS protected SMTP. If your email server does not provide SSL support, you can still secure it with SSL using a package like Stunnel (*http://www.stunnel.org/*). SSL can be used to effectively secure just about any service that runs over TCP.

SSH

Most people think of SSH as a secure replacement for **`telnet`**, just as **`scp`** and **`sftp`** are the secure counterparts of **`rcp`** and **`ftp`**. But SSH is much more than encrypted remote shell. Like SSL, it uses strong public key cryptography to verify the remote server and encrypt data. Instead of a PKI, it uses a key fingerprint cache that is checked before a connection is permitted. It can use passwords, public keys, or other methods for user authentication.

Many people do not know that SSH can also act as a general purpose encrypting tunnel, or even an encrypting web proxy. By first establishing an SSH connection to a trusted location near (or even on) a remote server, insecure protocols can be protected from eavesdropping and attack.

While this technique may be a bit advanced for many users, network architects can use SSH to encrypt traffic across untrusted links, such as wireless point-to-point links. Since the tools are freely available and run over standard TCP, any educated user can implement SSH connections for themselves, providing their own end-to-end encryption without administrator intervention.

OpenSSH (*http://openssh.org/*) is probably the most popular implementation on Unix-like platforms. Free implementations such as Putty (*http://www.putty.nl/*) and WinSCP (*http://winscp.net/*) are available for Windows. OpenSSH will also run on Windows under the Cygwin package (*http://www.cygwin.com/*). These examples will assume that you are using a recent version of OpenSSH.

To establish an encrypted tunnel from a port on the local machine to a port on the remote side, use the **-L** switch. For example, suppose you want to forward web proxy traffic over an encrypted link to the squid server at *squid.example.net*. Forward port 3128 (the default proxy port) using this command:

```
ssh -fN -g -L3128:squid.example.net:3128 squid.example.net
```

The **-fN** switches instruct ssh to fork into the background after connecting. The **-g** switch allows other users on your local segment to connect to the local machine and use it for encryption over the untrusted link. OpenSSH will use a public key for authentication if you have set one up, or it will prompt you for your password on the remote side. You can then configure your web browser to connect to localhost port 3128 as its web proxy service. All web traffic will then be encrypted before transmission to the remote side.

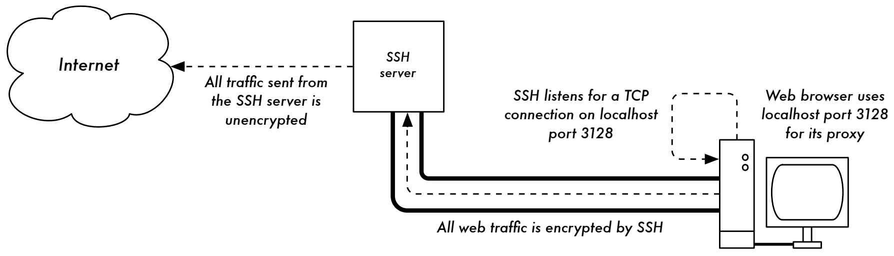
Internet
All traffic sent from the SSH server is unencrypted
SSH server
SSH listens for a TCP connection on localhost port 3128
Web browser uses localhost port 3128 for its proxy
All web traffic is encrypted by SSH

Figure 6.6: The SSH tunnel protects web traffic up to the SSH server itself.

SSH can also act as a dynamic SOCKS4 or SOCKS5 proxy. This allows you to create an encrypting web proxy, without the need to set up squid. Note that this is not a caching proxy; it simply encrypts all traffic.

```
ssh -fN -D 8080 remote.example.net
```

Configure your web browser to use SOCKS4 or SOCKS5 on local port 8080, and away you go.

SSH can encrypt data on any TCP port, including ports used for email. It can even compress the data along the way, which can decrease latency on low capacity links.

```
ssh -fNCg -L110:localhost:110 -L25:localhost:25 mailhost.example.net
```

The **-C** switch turns on compression. You can add as many port forwarding rules as you like by specifying the **-L** switch multiple times. Note that in order to bind to a local port less than 1024, you must have root privileges on the local machine.

These are just a few examples of the flexibility of SSH. By implementing public keys and using the ssh forwarding agent, you can automate the creation of encrypted tunnels throughout your wireless network, and protect your communications with strong encryption and authentication.

OpenVPN

OpenVPN is a free, open source VPN implementation built on SSL encryption. There are OpenVPN client implementations for a wide range of operating systems, including Linux, Windows 2000/XP and higher, OpenBSD, FreeBSD, NetBSD, Mac OS X, and Solaris. Being a VPN, it encapsulates all traffic (including DNS and all other protocols) in an encrypted tunnel, not just a single TCP port. Most people find it considerably easier to understand and configure than IPSEC.

OpenVPN also has some disadvantages, such as fairly high latency. Some amount of latency is unavoidable since all encryption/decryption is done in user space, but using relatively new computers on either end of the tunnel can minimize this. While it can use traditional shared keys, OpenVPN really shines when used with SSL certificates and a certificate authority. OpenVPN has many advantages that make it a good option for providing end-to-end security.

- It is based on a proven, robust encryption protocol (SSL and RSA)
- It is relatively easy to configure
- It functions across many different platforms
- It is well documented
- It's free and open source.

Like SSH and SSL, OpenVPN needs to connect to a single TCP port on the remote side. Once established, it can encapsulate all data down to the Networking layer, or even down to the Data-Link layer, if your solution requires it. You can use it to create robust VPN connections between individual machines, or simply use it to connect network routers over untrusted wireless networks.

VPN technology is a complex field, and is a bit beyond the scope of this section. It is important to understand how VPNs fit into the structure of your network in order to provide the best possible protection without opening up your organization to unintentional problems. There are many good on-line resources that deal with installing OpenVPN on a server and client, I recommend this article from Linux Journal: *http://www.linuxjournal.com/article/7949* as well as the official HOWTO: *http://openvpn.net/howto.html*

Tor & Anonymizers

The Internet is basically an open network based on trust. When you connect to a web server across the Internet, your traffic passes through many different routers, owned by a great variety of institutions, corporations and individuals. In principle, any one of these routers has the ability to look closely at your data, seeing as a minimum the source and destination addresses, and quite often also the actual content of the data. Even if your data is encrypted using a secure protocol, it is possible for your Internet provider to monitor the amount of data and the source and destination of that data. Often this is enough to piece together a fairly complete picture of your activities on-line.

Privacy and anonymity are important, and closely linked to each other. There are many valid reasons to consider protecting your privacy by ***anonymizing*** your network traffic. Suppose you want to offer Internet connectivity to your local community by setting up a number of access points for people to connect to. Whether you charge them for their access or not, there is always the risk that people use the network for something that is not legal in your country or region. You could plead with the legal system that this particular illegal action was not performed by yourself, but could have been performed by anyone connecting to your network. The problem is neatly sidestepped if it were technically infeasible to determine where your traffic was actually headed. And what about on-line censorship? Publishing web pages anonymously may also be necessary to avoid government censorship.

There are tools that allow you to anonymize your traffic in relatively easy ways. The combination of ***Tor*** (*http://tor.eff.org/*) and ***Privoxy*** (*http://www.privoxy.org/*) is a powerful way to run a local proxy server that will pass your Internet traffic through a number of servers all across the net, making it very difficult to follow the trail of information. Tor can be run on a local PC, under Microsoft Windows, Mac OSX, Linux and a variety of BSD's, where it anonymizes traffic from the browser on that particular machine. Tor and Privoxy can also be installed on a gateway server, or even a small embedded access point (such as a Linksys WRT54G) where they provides anonymity to all network users automatically.

Tor works by repeatedly bouncing your TCP connections across a number of servers spread throughout the Internet, and by wrapping routing information in a number of encrypted layers (hence the term ***onion routing***), that get peeled off as the packet moves across the network. This means that, at any given point in the network, the source and destination addresses cannot be linked together. This makes traffic analysis extremely difficult.

The need for the Privoxy privacy proxy in connection with Tor is due to the fact that name server queries (DNS queries) in most cases are not passed through the proxy server, and someone analyzing your traffic would easily be able to see that you were trying to reach a specific site (say *google.com*) by the fact that you sent a DNS query to translate google.com to the appropriate IP address. Privoxy connects to Tor as a SOCKS4a proxy, which uses hostnames (not IP addresses) to get your packets to the intended destination.

In other words, using Privoxy with Tor is a simple and effective way to prevent traffic analysis from linking your IP address with the services you use online. Combined with secure, encrypted protocols (such as those we have seen in this chapter), Tor and Privoxy provide a high level of anonymity on the Internet.

Monitoring

Computer networks (and wireless networks in particular) are incredibly entertaining and useful inventions. Except, of course, when they don't work. Your users may complain that the network is "slow" or "broken", but what does that really mean? Without insight into what is actually happening, administering a network can be very frustrating.

In order to be an effective network administrator, you need access to tools that show you exactly what is happening on your network. There are several different classes of monitoring tools. Each shows you a different aspect of what is "going on", from the physical radio interaction all the way to how user applications interact with each other. By watching how the network performs over time, you can get an idea of what is "normal" for your network, and even be notified automatically when things seem to be out of the ordinary. The tools listed in this section are all quite powerful, and are freely available for download from the sources listed.

Network detection

The simplest wireless monitoring tools simply provide a list of available networks, along with basic information (such as signal strength and channel). They let you quickly detect nearby networks and determine if they are in range or are causing interference.

- **The built-in client.** All modern operating systems provide built-in support for wireless networking. This typically includes the ability to scan for available networks, allowing the user to choose a network from a list. While virtually all wireless devices are guaranteed to have a simple scanning utility, functionality can vary widely between implementations. These tools are typically only useful for configuring a computer in a home or office setting. They tend to provide little information apart from network names and the available signal to the access point currently in use.
- **Netstumbler** (*http://www.netstumbler.com/*). This is the most popular tool for detecting wireless networks using Microsoft Windows. It supports a variety of wireless cards, and is very easy to use. It will detect open and encrypted networks, but cannot detect "closed" wireless networks. It also features a signal/noise meter that plots radio receiver data as a graph over time. It also integrates with a variety of GPS devices, for logging precise location and signal strength information. This makes Netstumbler a handy tool to have for an informal site survey.

- **Ministumbler** (*http://www.netstumbler.com/*). From the makers of Netstumbler, Ministumbler provides much of the same functionality as the Windows version, but works on the Pocket PC platform. Ministumbler is handy to run on a handheld PDA with a wireless card for detecting access points in the field.
- **Macstumbler** (*http://www.macstumbler.com/*). While not directly related to the Netstumbler, Macstumbler provides much of the same functionality but for the Mac OS X platform. It works with all Apple Airport cards.
- **Wellenreiter** (*http://www.wellenreiter.net/*). Wellenreiter is a nice graphical wireless network detector for Linux. It requires Perl and GTK, and supports Prism2, Lucent, and Cisco wireless cards.

Protocol analyzers

Network protocol analyzers provide a great deal of detail about information flowing through a network, by allowing you to inspect individual packets. For wired networks, you can inspect packets at the data-link layer or above. For wireless networks, you can inspect information all the way down to individual 802.11 frames. Here are several popular (and free) network protocol analyzers:

- **Ethereal** (*http://www.ethereal.com/*). Ethereal is probably the most popular protocol analyzer available. It works with Linux, Windows, Mac OS X, and the various BSD systems. Ethereal will capture packets directly "from the wire" and display them in an intuitive graphical interface. It can decode over 750 different protocols, everything from 802.11 frames to HTTP packets. It will reassemble fragmented packets and follow entire TCP sessions easily, even if other data has broken up the sample. Ethereal is very valuable for troubleshooting tricky network problems, and figuring out exactly what is happening when two computers converse "on the wire".
- **Kismet** (*http://www.kismetwireless.net/*). Kismet is a powerful wireless protocol analyzer for Linux, Mac OS X, and even the embedded OpenWRT Linux distribution. It works with any wireless card that supports passive monitor mode. In addition to basic network detection, Kismet will passively log all 802.11 frames to disk or to the network in standard PCAP format, for later analysis with tools like Ethereal. Kismet also features associated client information, AP hardware fingerprinting, Netstumbler detection, and GPS integration.

  Since it is a passive network monitor, it can even detect "closed" wireless networks by analyzing traffic sent by wireless clients. You can run Kismet on several machines at once, and have them all report over the network back to a central user interface. This allows for wireless monitoring over a large area, such as a university or corporate campus. Since it uses the passive monitor mode, it does all of this without transmitting any data.

- **KisMAC** (*http://kismac.binaervarianz.de/*). Exclusively for the Mac OS X platform, KisMAC does much of what Kismet can do, but with a slick Mac OS X graphical interface. It is a passive scanner that will log data to disk in PCAP format compatible with Ethereal. It does not support passive scanning with AirportExtreme cards (due to limitations in the wireless driver), but it supports passive mode with a variety of USB wireless cards.
- **Driftnet** and **Etherpeg**. These tools decode graphical data (such as GIF and JPEG files) and display them as a collage. As mentioned earlier, tools such as these are of limited use in troubleshooting problems, but are very valuable for demonstrating the insecurity of unencrypted protocols. Etherpeg is available from *http://www.etherpeg.org/*, and Driftnet can be downloaded at *http://www.ex-parrot.com/~chris/driftnet/*.

Bandwidth monitoring

The network is slow. Who is hogging all of the bandwidth? By using a good bandwidth monitoring tool, you can easily determine the source of spam and virus flooding problems. Such tools can also help you to plan for future capacity as the network users outgrow the available pipe. These tools will give you a visual representation of how traffic is flowing throughout your network, including traffic coming from a particular machine or service.

- **MRTG** (*http://people.ee.ethz.ch/~oetiker/webtools/mrtg/*). Most network administrators have encountered MRTG at some point in their travels. Originally written in 1995, MRTG is possibly the most widely used bandwidth monitoring application. Using Perl and C, it builds a web page full of graphs detailing the inbound and outbound traffic used on a particular network device. MRTG makes it simple to query network switches, access points, servers, and other devices and display the results as graphs that change over time.
- **RRDtool** (*http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/*). Developed by the same people who wrote mrtg, rrdtool is a more powerful generic monitoring application. RRD is short for “round-robin database”. It is a generic data format that allows you to easily track any particular data point as a set averaged over time. While rrdtool does not directly monitor interfaces or devices, many monitoring packages rely on it to store and display the data they collect. With a few simple shell scripts, you can easily monitor your network switches and access points, and plot the bandwidth used as a graph on a web page.
- **ntop** (*http://www.ntop.org/*). For historical traffic analysis and usage, you will want to investigate ntop. This program builds a detailed real-time report on observed network traffic, displayed in your web browser. It inte-

grates with rrdtool, and makes graphs and charts visually depicting how the network is being used. On very busy networks, ntop can use a lot of CPU and disk space, but it gives you extensive insight into how your network is being used. It runs on Linux, BSD, Mac OS X, and Windows.

- **iptraf** (*http://iptraf.seul.org/*). If you need to instantly take a snapshot of network activity on a Linux system, give iptraf a try. It is a command-line utility that gives you an up-to-the-second look at connections and network flows, including ports and protocols. It can be very handy for determining who is using a particular wireless link, and how heavily it is loaded. For example, by showing the detailed statistical breakdown for an interface, you can instantly find peer-to-peer client users, and determine exactly how much bandwidth they are currently using.

Troubleshooting

What do you do when the network breaks? If you can't access a web page or email server, and clicking the reload button doesn't fix the problem, then you'll need to be able to isolate the exact location of the problem. These tools will help you to determine just where a connection problem exists.

- **ping**. Just about every operating system (including Windows, Mac OS X, and of course Linux and BSD) includes a version of the ping utility. It uses ICMP packets to attempt to contact a specified host, and tells you how long it takes to get a response.

  Knowing what to ping is just as important as knowing how to ping. If you find that you cannot connect to a particular service in your web browser (say, *http://yahoo.com/*), you could try to ping it:

```
$ ping yahoo.com
PING yahoo.com (66.94.234.13): 56 data bytes
64 bytes from 66.94.234.13: icmp_seq=0 ttl=57 time=29.375 ms
64 bytes from 66.94.234.13: icmp_seq=1 ttl=56 time=35.467 ms
64 bytes from 66.94.234.13: icmp_seq=2 ttl=56 time=34.158 ms
^C
--- yahoo.com ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 29.375/33.000/35.467/2.618 ms
```

  Hit control-C when you are finished collecting data. If packets take a long time to come back, there may be network congestion. If return ping packets have an unusally low ttl, you may have routing problems between your machine and the remote end. But what if the ping doesn't return any data at all? If you are pinging a name instead of an IP address, you may be running into DNS problems.

Try pinging an IP address on the Internet. If you can't reach it, it's a good idea to see if you can ping your default router:

```
$ ping 216.231.38.1
PING 216.231.38.1 (216.231.38.1): 56 data bytes
64 bytes from 216.231.38.1: icmp_seq=0 ttl=126 time=12.991 ms
64 bytes from 216.231.38.1: icmp_seq=1 ttl=126 time=14.869 ms
64 bytes from 216.231.38.1: icmp_seq=2 ttl=126 time=13.897 ms
^C
--- 216.231.38.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 12.991/13.919/14.869/0.767 ms
```

If you can't ping your default router, then chances are you won't be able to get to the Internet either. If you can't even ping other IP addresses on your local LAN, then it's time to check your connection. If you're using Ethernet, is it plugged in? If you're using wireless, are you connected to the proper wireless network, and is it in range?

Network debugging with ping is a bit of an art, but it is useful to learn. Since you will likely find ping on just about any machine you will work on, it's a good idea to learn how to use it well.

- **traceroute** and **mtr** (*http://www.bitwizard.nl/mtr/*). As with ping, traceroute is found on most operating systems (it's called **tracert** in some versions of Microsoft Windows). By running traceroute, you can find the location of problems between your computer and any point on the Internet:

```
$ traceroute -n google.com
traceroute to google.com (72.14.207.99), 64 hops max, 40 byte packets
 1  10.15.6.1  4.322 ms  1.763 ms  1.731 ms
 2  216.231.38.1  36.187 ms  14.648 ms  13.561 ms
 3  69.17.83.233  14.197 ms  13.256 ms  13.267 ms
 4  69.17.83.150  32.478 ms  29.545 ms  27.494 ms
 5  198.32.176.31  40.788 ms  28.160 ms  28.115 ms
 6  66.249.94.14  28.601 ms  29.913 ms  28.811 ms
 7  172.16.236.8  2328.809 ms  2528.944 ms  2428.719 ms
 8  * * *
```

The **-n** switch tells traceroute not to bother resolving names in DNS, and makes the trace run more quickly. You can see that at hop seven, the round trip time shoots up to more than two seconds, while packets seem to be discarded at hop eight. This might indicate a problem at that point in the network. If this part of the network is in your control, it might be worth starting your troubleshooting effort there.

My TraceRoute (mtr) is a handy program that combines ping and traceroute into a single tool. By running mtr, you can get an ongoing average of latency and packet loss to a single host, instead of the momentary snapshot that ping and traceroute provide.

```
                             My traceroute  [v0.69]
tesla.rob.swn (0.0.0.0)        (tos=0x0 psize=64 bitpatSun Jan  8 20:01:26 2006
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                       Packets               Pings
 Host                                Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. gremlin.rob.swn                   0.0%     4    1.9   2.0   1.7   2.6   0.4
 2. er1.sea1.speakeasy.net            0.0%     4   15.5  14.0  12.7  15.5   1.3
 3. 220.ge-0-1-0.cr2.sea1.speakeasy.  0.0%     4   11.0  11.7  10.7  14.0   1.6
 4. fe-0-3-0.cr2.sfo1.speakeasy.net   0.0%     4   36.0  34.7  28.7  38.1   4.1
 5. bas1-m.pao.yahoo.com              0.0%     4   27.9  29.6  27.9  33.0   2.4
 6. so-1-1-0.pat1.dce.yahoo.com       0.0%     4   89.7  91.0  89.7  93.0   1.4
 7. ae1.p400.msr1.dcn.yahoo.com       0.0%     4   91.2  93.1  90.8  99.2   4.1
 8. ge5-2.bas1-m.dcn.yahoo.com        0.0%     4   89.3  91.0  89.3  93.4   1.9
 9. w2.rc.vip.dcn.yahoo.com           0.0%     3   91.2  93.1  90.8  99.2   4.1
```

The data will be continuously updated and averaged over time. As with ping, you should hit control-C when you are finished looking at the data. Note that you must have root privileges to run mtr.

While these tools will not revel precisely what is wrong with the network, they can give you enough information to know where to continue troubleshooting.

Throughput testing

How fast can the network go? What is the actual usable capacity of a particular network link? You can get a very good estimate of your throughput capacity by flooding the link with traffic and measuring how long it takes to transfer the data. While there are web pages available that will perform a "speed test" in your browser (such as *http://www.dslreports.com/stest*), these tests are increasingly inaccurate as you get further from the testing source. Even worse, they do not allow you to test the speed of a particular link, but only the speed of your link to the Internet. Here are two tools that will allow you to perform throughput testing on your own networks.

- **ttcp** (*http://ftp.arl.mil/ftp/pub/ttcp/*). Now a standard part of most Unix-like systems, ttcp is a simple network performance testing tool. One instance is run on either side of the link you want to test. The first node runs in receive mode, and the other transmits:

```
node_a$ ttcp -r -s

node_b$ ttcp -t -s node_a
ttcp-t: buflen=8192, nbuf=2048, align=16384/0, port=5001  tcp -> node_a
ttcp-t: socket
```

```
ttcp-t: connect
ttcp-t: 16777216 bytes in 249.14 real seconds = 65.76 KB/sec +++
ttcp-t: 2048 I/O calls, msec/call = 124.57, calls/sec = 8.22
ttcp-t: 0.0user 0.2sys 4:09real 0% 0i+0d 0maxrss 0+0pf 7533+0csw
```

After collecting data in one direction, you should reverse the transmit and receive partners to test the link in the other direction. It can test UDP as well as TCP streams, and can alter various TCP parameters and buffer lengths to give the network a good workout. It can even use a user-supplied data stream instead of sending random data. Remember that the speed readout is in kilobytes, not kilobits. Multiply the result by 8 to find the speed in kilobits per second.

The only real disadvantage to ttcp is that it hasn't been developed in years. Fortunately, the code has been released in the public domain and is freely available. Like ping and traceroute, ttcp is found as a standard tool on many systems.

- **iperf** (*http://dast.nlanr.net/Projects/Iperf/*). Much like ttcp, iperf is a commandline tool for estimating the throughput of a network connection. It supports many of the same features as ttcp, but uses a "client" and "server" model instead of a "receive" and "transmit" pair. To run iperf, launch a server on one side and a client on the other:

```
node_a$ iperf -s

node_b$ iperf -c node_a
------------------------------------------------------------
Client connecting to node_a, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  5] local 10.15.6.1 port 1212 connected with 10.15.6.23 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-11.3 sec   768 KBytes   558 Kbits/sec
```

The server side will continue to listen and accept client connections on port 5001 until you hit control-C to kill it. This can make it handy when running multiple test runs from a variety of locations.

The biggest difference between ttcp and iperf is that iperf is under active development, and has many new features (including IPv6 support). This makes it a good choice as a performance tool when building new networks.

Network health

By tracking information over time, you can get an overall idea of the general health of the network and its services. These tools will show you network trends and even notify a human when problems present themselves. More often than not, the systems will notice trouble before a person has a chance to call tech support.

- **cacti** (*http://www.cacti.net/*). As mentioned earlier, many tools use RRDtool as a back-end to build graphs for data that they collect. Cacti is such a tool. It is a PHP-based network management tool that simplifies data gathering and graph generation. It stores its configuration in a MySQL database, and is integrated with SNMP. This makes it very straightforward to map out all of the devices on your network, and monitor everything from network flows to CPU load. Cacti has an extensible data collection scheme that lets you collect just about any kind of data you can think of (such as radio signal, noise, or associated users) and plot it on a graph over time. Thumbnail views of your graphs can be combined into a single web page. This lets you observe the overall state of your network at a glance.
- **SmokePing** (*http://people.ee.ethz.ch/~oetiker/webtools/smokeping/*). Yet another tool by Tobias Oetiker, SmokePing is a tool written in Perl that shows packet loss and latency on a single graph. It is very useful to run SmokePing on a host with good connectivity to your entire network. Over time, trends are revealed that can point to all sorts of network problems. Combined with MRTG or Cacti, you can observe the effect that network congestion has on packet loss and latency. SmokePing can optionally send alerts when certain conditions are met, such as when excessive packet loss is seen on a link for an extended period of time.
- **Nagios** (*http://www.nagios.org/*). Nagios is a service monitoring tool. In addition to tracking the performance of simple pings (as with SmokePing), Nagios can watch the performance of actual services on any number of machines. For example, it can periodically query your web server, and be sure that it returns a valid web page. If a check should fail, Nagios can notify a person or group via email, SMS, or IM.

  While Nagios will certainly help a single admin to monitor a large network, Nagios is best used when you have a troubleshooting team with responsibilities divided between various members. Trouble events can be configured to ignore transient problems, then escalate notifications only to people who are responsible for fixing them. If the problem goes on for a predefined period of time without being acknowledged, other people can additionally be notified. This allows temporary problems to be simply logged without bothering people, and for real problems to be brought to the attention of the team.