CHAPTER

21

Network Attack and Defense

Whoever thinks his problem can be solved using cryptography, doesn't understand his problem and doesn't understand cryptography.

— Attributed by Roger Needham and Butler Lampson to Each Other

If you spend more on coffee than on IT security, then you will be hacked. What's more, you deserve to be hacked.

— Richard Clarke, Former U.S. Cybersecurity Tsar

21.1 Introduction

So far we've seen a large number of attacks against individual computers and other devices. But attacks increasingly depend on connectivity. Consider the following examples.

1. An office worker clicks on an attachment in email. This infects her PC with malware that compromises other machines in her office by snooping passwords that travel across the LAN.
2. The reason she clicked on the attachment is that the email came from her mother. The malware had infected her mother's machine and then sent out a copy of a recent email, with itself attached, to everyone in mum's address book.
3. Her mother in turn got infected by an old friend who chose a common password for his ISP account. When there are many machines on a network, the bad guys don't have to be choosy; rather than trying to guess the password for a particular account, they just try one password over and over for millions of accounts. Given a webmail account, they can send out bad email to the whole contact list.

4. Another attack technique that makes sense only in a network context is *Google hacking*. Here, the bad guys use search engines to find web servers that are running vulnerable applications.
5. The malware writers infect a whole lot of PCs more or less at random using a set of tricks like these. They then look for choice pickings, such as machines in companies from which large numbers of credit card numbers can be stolen, or web servers that can be used to host phishing web pages as well. These may be auctioned off to specialists to exploit. Finally they sell on the residual infected machines for under a dollar a time to a *botnet herder* — who operates a large network of compromised machines that he rents out to spammers, phishermen and extortionists.
6. One of the applications is *fast-flux*. This changes the IP address of a web site perhaps once every 20 minutes, so that it's much more difficult to take down. A different machine in the botnet acts as the host (or as a proxy to the real host) with each change of IP address, so blocking such an address has at most a temporary effect. Fast-flux hosting is used by the better phishing gangs for their bogus bank websites.

There are many attacks, and defenses, that emerge once we have large numbers of machines networked together. These depend on a number of factors, the most important of which are the protocols the network uses. A second set of factors relate to the *topology of the network*: is every machine able to contact every other machine, or does it only have direct access to a handful of others? In our example above, a virus spreads itself via a social network — from one friend to another, just like the flu virus.

I've touched on network aspects of attack and defense before, notably in the chapters on telecomms and electronic warfare. However in this chapter I'm going to try to draw together the network aspects of security in a coherent framework. First I'm going to discuss networking protocols, then malware; then defensive technologies, from filtering and intrusion detection to the widely-used crypto protocols TLS, SSH, IPsec and wireless LAN encryption. Finally I'll discuss network topology. The most immediate application of this bundle of technologies is the defence of networks of PCs against malware; however as other devices go online the lessons will apply there too. In addition, many network security techniques can be used for multiple purposes. If you invent a better firewall, then — like it or not — you've also invented a better machine for online censorship and a better police wiretap device as well. Conversely, if mobility and virtual private networks make life tough for the firewall designer, they can give the censor and the police wiretap department a hard time, too.

21.2 Vulnerabilities in Network Protocols

This book isn't an appropriate place to explain basic network protocols. The telegraphic summary is as follows. The *Internet Protocol* (IP) is a stateless protocol that transfers packet data from one machine to another; IP version 4 uses 32-bit *IP addresses*, often written as four decimal numbers in the range 0–255, such as 172.16.8.93. People have started to migrate to IP version 6, as the 4 billion possible IPv4 addresses will have been allocated sometime between 2010 and 2015; IPv6 uses 128-bit addresses. Most modern kit is ready to use IPv6 but the changeover, which companies will probably do one LAN at a time, will no doubt throw up some interesting problems. The *Domain Name System* (DNS) allows mnemonic names such as `www.ross-anderson.com` to be mapped to IP addresses of either kind; there's a hierarchy of DNS servers that do this, ranging from thirteen top-level servers down through machines at ISPs and on local networks, which cache DNS records for performance and reliability.

The core routing protocol of the Internet is the *Border Gateway Protocol* (BGP). The Internet consists of a large number of *Autonomous Systems* (ASs) such as ISPs, telcos and large companies, each of which controls a range of IP addresses. The routers — the specialized computers that ship packets on the Internet — use BGP to exchange information about what routes are available to get to particular blocks of IP addresses, and to maintain routing tables so they can select efficient routes.

Most Internet services use a protocol called *transmission control protocol* (TCP) that is layered on top of IP and provides virtual circuits. It does this by splitting up the data stream into IP packets and reassembling it at the far end, automatically retransmitting any packets whose receipt is not acknowledged. IP addresses are translated into the familiar Internet host addresses using the *domain name system* (DNS), a worldwide distributed service in which higher-level name servers point to local name servers for particular domains. Local networks mostly use ethernet, in which devices have unique ethernet addresses (also called MAC addresses) that are mapped to IP addresses using the *address resolution protocol* (ARP). Because of the growing shortage of IP addresses, most organisations and ISPs now use the *Dynamic Host Configuration Protocol* (DHCP) to allocate IP addresses to machines as needed and to ensure that each IP address is unique. So if you want to track down a machine that has done something wicked, you will often have to get the logs that map MAC addresses to IP addresses.

There are many other components in the protocol suite for managing communications and providing higher-level services. Most of them were developed in the good old days when the net had only trusted hosts and security wasn't a concern. So there is little authentication built in. This is

a particular problem with DNS and with BGP. For example, if a small ISP mistakenly advertises to a large neighbour that it has good routes to a large part of the Internet, it may be swamped by the traffic. Here at least there are sound economic incentives, in that ISPs either swap, or pay for, routes with their peers; BGP security is in effect bodged up using manual intervention. DNS is trickier, in that disruptions are often malicious rather than mistaken. A DNS server may be fed wrong information to drive clients to a wicked website. This can be done either wholesale, by an attack on the DNS servers of a large ISP, or at the local level. For example, many homes have a wireless router attached to a broadband connection, and the router contains the address of the DNS server the customer uses. In an attack called *Drive-By Pharming*, the villain lures you to view a web page containing javascript code that sets your router's DNS server to one under his control [1213]. The effect is that next time you try to go to `www.citibank.com`, you may be directed to a phishing site that emulates it. For this reason it's a really good idea to change the default password on your home router.

21.2.1 Attacks on Local Networks

Suppose the attacker controls one of your PCs. Perhaps one of your employees was careless; or maybe he's gone bad, and wants to take over an account in someone else's name to defraud you, or to do some other bad thing such as downloading child porn in the hope of framing someone. There are several possibilities open to him.

1. He can install packet sniffer software to harvest passwords, get the root password, and thus take over a suitable account. Password-sniffing attacks can be blocked if you use challenge-response password generators, or a protocol such as Kerberos or ssh to ensure that clear text passwords don't go over the LAN. I described Kerberos in Chapter 3, and I'll describe SSH later.
2. Another approach is to masquerade as a machine where the target user — say the sysadmin — has already logged on. It is often possible for the attacker simply to set his MAC address and IP address to those of the target. In theory, the target machine should send 'reset' packets when it sees traffic to its IP address that's not in response to its own packets; but many machines nowadays have personal firewalls, which throw away 'suspicious' packets. As a result, the alarm doesn't get raised [300].
3. There's a whole host of technical address-hijacking attacks that work fine against old-fashioned LANs. An example I gave in the first edition of my book was that the attacker gives wrong answers to ARP messages, claiming to be the target, and may stop the target machine noticing and

raising the alarm by sending it a false subnet mask. Another possibility is to send bogus DHCP messages. Attacks like this may or may not work against a modern switched ethernet, depending on how it's configured.

4. A further set of attacks target particular platforms. For example, if the target company uses Linux or Unix servers, they are likely to use Sun's *Network File System* (NFS) for file sharing. This allows workstations to use a network disk drive as if it were a local disk, and has a number of well-known vulnerabilities to attackers on the same LAN. When a volume is first mounted, the client gets a *root filehandle* from the server. This is in effect an access ticket that refers to the root directory of the mounted filesystem, but that doesn't depend on the time, or the server generation number, and can't be revoked. There is no mechanism for per-user authentication: the server must trust a client completely or not at all. Also, NFS servers often reply to requests from a different network interface to the one on which the request arrived. So it's possible to wait until an administrator is using a file server and then masquerade as him to overwrite the password file. Filehandles can also be intercepted by network sniffing, though again, switched ethernet makes this harder. Kerberos can be used to authenticate clients and servers, but many firms don't use it; getting it to work in a heterogeneous environment can be difficult.

So the ease with which a bad machine on your network can take over other machines depends on how tightly you have the network locked down, and the damage that a bad machine can do will depend on the size of the local network. There are limits to how far a sysadmin can go; your firm might need to run a complex mixture of legacy systems for which Kerberos just can't be got to work. Also, a security-conscious system administrator can impose real costs. At our lab we argued with our sysadmins for years, trying to get access to the Internet for visiting guests, while they resisted on both technical protection and policy grounds (our academic network shouldn't be made available to commercial users). In the end, we solved the problem by setting up a separate guest network that is connected to a commercial ISP rather than to the University's backbone.

This raises a wider problem: where's the network boundary? In the old days, many companies had a single internal network, connected to the Internet via a firewall of some kind. But compartmentation often makes sense, as I discussed in Chapter 9: separate networks for each department can limit the damage that a compromised machine can do. There may be particularly strong arguments for this if some of your departments may have high protection requirements, while others need great flexibility. In our university, for example, we don't want the students on the same LAN that the payroll folks use; in fact we separate student, staff and administrative networks, and the first two of these

are also separated by college and department. Recently, mobility and virtual networks have made definition of clear network boundaries even harder. This debate goes by the buzz-word of *deperimiterisation* and I'll return to it later.

One final attack is worth mentioning under the heading of attacks on local networks, and that's the *rogue access point*. Occasionally one finds WiFi access points in public areas, such as airports, that have been deployed maliciously. The operator might sit in the airport lounge with a laptop that accesses the Internet via a paid WiFi service and advertises a free one; if you use it, he'll be able to sniff any plaintext passwords you enter, for example to your webmail or Amazon account, and if you tried to do online banking he might conceivably send you to a malicious site. So the effects can be somewhat like drive-by pharming, although more reliable and less scalable. In addition, rogue access points may also be devices that employees have installed for their own convenience in defiance of corporate policy, or even official nodes that have been misconfigured so that they don't encrypt the traffic. Whether unencrypted WiFi traffic is a big deal will depend on the circumstances; I'll discuss this in more detail later when we come to encryption.

21.2.2 Attacks Using Internet Protocols and Mechanisms

Moving up now to the Internet protocol suite, the basic problem is similar: there is no real authenticity protection in the default mechanisms. This is particularly manifest at the lower level TCP/IP protocols, and has given rise to many attacks.

Consider for example the 3-way handshake used by Alice to initiate a TCP connection to Bob and set up sequence numbers.

This protocol can be exploited in a surprising number of different ways. Now that service denial is becoming really important, let's start off with the simplest service denial attack: the *SYN flood*.

21.2.2.1 SYN Flooding

The attack is quite simply to send a large number of SYN packets and never acknowledge any of the replies. This leads the recipient (Bob, in Figure 21.1) to accumulate more records of SYN packets than his software can handle. This attack had been known to be theoretically possible since the 1980s but came to public attention when it was used to bring down Panix, a New York ISP, for several days in 1996.

A technical fix has been incorporated in Linux and some other systems. This is the so-called 'SYNcookie'. Rather than keeping a copy of the incoming SYN packet, *B* simply sends out as *Y* an encrypted version of *X*. That way, it's

A → B: SYN; my number is X
B → A: ACK; now X+1
SYN; my number is Y
A → B: ACK; now Y+1
(start talking)

Figure 21.1: TCP/IP handshake

not necessary to retain state about sessions which are half-open. Despite this, SYN floods are still a big deal, accounting for the largest number of reported attacks (27%) in 2006, although they were only the third-largest in terms of traffic volume (18%, behind UDP floods and application-layer attacks) [86].

There is an important general principle here: when you're designing a protocol that anyone can invoke, don't make it easy for malicious users to make honest ones consume resources. Don't let anyone in the world force your software to allocate memory, or do a lot of computation. In the online world, that's just asking for trouble.

21.2.2.2 Smurfing

A common way of bringing down a host in the 90s was *smurfing*. This exploited the *Internet control message protocol* (ICMP), which enables users to send an echo packet to a remote host to check whether it's alive. The problem was with broadcast addresses that are shared by a number of hosts. Some implementations of the Internet protocols responded to pings to both the broadcast address as well as the local address — so you could test a LAN to see what was alive. A collection of such hosts at a broadcast address is called a *smurf amplifier*. Bad guys would construct a packet with the source address forged to be that of the victim, and send it to a number of smurf amplifiers. These would then send a flurry of packets to the target, which could swamp it. Smurfing was typically used by kids to take over an *Internet relay chat* (IRC) server, so they could assume control of the chatroom. For a while this was a big deal, and the protocol standards were changed in August 1999 so that ping packets sent to a broadcast address are no longer answered [1144]. Another part of the fix was socio-economic: vigilante sites produced lists of smurf amplifiers. Diligent administrators spotted their networks on there and fixed them; the lazy ones then found that the bad guys used more and more of their bandwidth, and thus got pressured into fixing the problem too. By now (2007), smurfing is more or less fixed; it's no longer an attack that many people use.

But there's a useful moral: *don't create amplifiers*. When you design a network protocol, be extremely careful to ensure that no-one who puts one packet in can get two packets out. It's also important to avoid feedback and loops. A classic example was *source routing*. A feature of early IP that enabled the sender

of a packet to specify not just its destination but the route that it should take. This made attacks too easy: you'd just send a packet from A to B to C to B to C and so on, before going to its final destination. Most ISPs now throw away all packets with source routing set. (There was an alarm in early 2007 when it turned out that source routing had found its way back into the specification for IPv6, but that's now been fixed [417].)

21.2.2.3 Distributed Denial of Service Attacks

As the clever ways of creating service-denial attacks have been closed off one by one, the bad guys have turned increasingly to brute force, for example by sending floods of UDP packets from infected machines. The *distributed denial of service* (DDoS) attack made its appearance in October 1999 with the attack already mentioned on a New York ISP, Panix. In DDoS, the attacker subverts a large number of machines over a period of time and, or on a given signal, these machines all start to bombard the target with traffic [391]. Curiously, most of the machines in the first botnets around 1999–2000 were U.S. medical sites. The FDA insisted that medical Unix machines which were certified for certain clinical uses have a known configuration. Once bugs were found in this, there was a guaranteed supply of vulnerable machines; an object lesson in the dangers of monoculture.

Nowadays, botnets are assembled using all sorts of vulnerabilities, and a market has arisen whereby people who specialise in hacking machines can sell their product to people who specialise in herding them and extracting value. Compromised machines typically pass down a kind of value chain; they are first used for targeted attacks, then for sending spam, then (once they get known to spam filters) for applications like fast flux, and then finally (once they're on all the blacklists) for DDoS.

DDoS attacks have been launched at a number of high-profile web sites, including Amazon and Yahoo, but nowadays the major sites have so much bandwidth that they're very hard to dent. The next development was extortionists taking out online horserace-betting sites just before popular race meetings that would have generated a lot of business, and demanding ransoms not to do it again. Some bookmakers moved their operations to high-bandwidth hosting services such as Akamai that are highly distributed and can cope with large packet volumes, and others to specialist ISPs with packet-washing equipment that filters out bad packets at high speed. However the real fix for extortion wasn't technical. First, the bookmakers got together, compared notes, and resolved that in future none of them would pay any ransom. Second, the Russian government was leant on to deal with the main gang; three men were arrested in 2004 and sent to prison for eight years in 2006 [791].

For a while, there was a technical arms race. Attackers started to spoof source IP addresses, and to reflecting packets off innocuous hosts [1011]. One

countermeasure was traceback: the idea was that whenever a router forwards a packet, it would also send an ICMP packet to the destination, with a probability of about 1 in 20,000, containing details of the previous hop, the next hop, and as much of the packet as will fit. Large-scale flooding attacks could thus be traced back to the responsible machines, even despite forged source IP addresses [154]. However, this arms race has largely fizzled out, and for a number of reasons. First, Microsoft changed their network stack to make it much harder for an infected machine to send a packet with a spoofed IP address; you now need to hack the operating system, not just any old application. Second, more and more equipment at the edges of the network won't accept spoofed packets, and now about half of broadband ISPs filter them out [86].

However, the main reasons for the arms race stopping had to do with economics. Tracing back packets didn't work across different autonomous systems; if you were a large telco, for example, you would not give network access to your competitors' technical staff. So the bad guys found that in practice nobody came after them, and stopped using spoofing. Also, once markets emerged round about 2004 for botnet machines to be bought and sold (along with credit card numbers, spam contracts and other criminal services), the price of compromised machines fell so low, and botnets started to become so large, that the whole game changed. Instead of using a handful of compromised machines to send out clever attacks via amplifiers using spoofed source addresses, the bad guys simply burn thousands of end-of-life botnet machines to send the bad packets directly. The rapier has been replaced with the Kalashnikov.

Most recently, in 2005–7, there have been attempts to target core services such as DNS and thus take down the whole Internet. DNS has now been expanded to thirteen servers (the maximum the protocol will support), and many of them use *anycast* — a protocol whereby when you ask to resolve the domain name of a host into an IP address, the result that you get depends on where you are. The net effect is that DNS has become a massively distributed system using a lot of very fast machines connected to very high-capacity networks. In the end, the brute force of the modern DDoS attack was simply answered by even more brute force. If a hundred peasants with Kalashnikovs are going to shoot at you, you'd better buy a tank with good enough armor to absorb the fire.

Large-scale DDoS attacks on critical services seem quiescent at present. There are a few residual worries, though.

- There's a rising tide of DDoS attacks that happen by accident rather than as a result of malice. For example, in 2003 the University of Wisconsin-Madison found itself receiving hundreds of thousands of packets per second requesting the time. It turned out that Netgear had sold some 700,000 routers that were hard-coded to ask their time server what time

it was, and to ask again a second later if no response was received. Netgear ended up paying the university to maintain a high-bandwidth time server for them. There have been dozens of similar incidents [303].

- There's a steady stream of DDoS attacks by spammers and phishermen on the websites of organisations that try to hinder their activities, such as Artists Against 419 and Spamhaus.
- There are continuing worries that DDoS attacks might come back on an even larger scale. As of September 2007, there are several botnets with over half a million machines [742]. The operators of such networks can send out packet floods that will disable all but the biggest sites. These worries have been amplified in some quarters by the 2007 attacks on Estonia (even although that attack would not have harmed a large commercial target like Microsoft or Google). The highest attack rate seen in 2006 was 24 Gbit/sec, compared with 10 Gbit/sec in 2004 and 1.2 Gbit/sec in 2002 [86]. A further order-of-magnitude increase could put all but the most distributed targets at risk.
- Even if we never see the Internet taken down by a monster botnet, attacks can still be carried out on smaller targets. Prior to the Estonia incident, there had been a DDoS attack on the servers of an opposition party in Kyrgyzstan, and these followed the site when it was relocated to North America [1081]. Certainly, DDoS puts a weapon in the hands of gangsters that can be rented out to various unsavoury people.
- That said, one mustn't forget online activism. If a hundred thousand people send email to the White House protesting against some policy or other, is this a DDoS attack? Protesters should not be treated as felons; but drawing legislative distinctions can be hard.

21.2.2.4 Spam

Spam is in some respects similar to a DDoS attack: floods of generally unwanted traffic sent out for the most part by botnets, and often with clear criminal intent. The technical aspects are related, in that both email and the web protocols (`smtp` and `http`) assume wrongly that the lower levels are secure. Just as DDoS bots may forge IP addresses, spam bots may forge the sender's email address.

Spam differs in a various ways, though, from packet-level DDoS. First, it's enough of an annoyance and a cost for there to be real pressure on ISPs to send less of it. If you're a medium-sized ISP you will typically peer with other ISPs as much as you can, and buy routes from large telcos only where you can't trade for them; if other ISPs see you as a source of spam they may drop your peering arrangements, which costs real money. As a result, some ISPs are starting to do *egress filtering*: they monitor spam coming out of their own networks and then quarantine the infected PCs into a 'walled garden' from which they have access to antivirus software but little else [300].

Second, unlike DDoS, spam does not seem to be tailing off. It does appear that spammers are consolidating, in that most spam comes from several dozen large gangs. This is apparent from the 'lumpiness' of spam statistics: if there were hundreds of thousands of mom-and-pop spam operations, you'd expect to see spam volumes pretty constant, but this is no longer what we see [305]. So rather than spending more money on spam filters, it might be cheaper to get the police to arrest the gangs. Trends do change over time, though. Between 2006 and 2007, we've seen a drop in *backscatter* — in messages sent to the people whose email addresses were forged by spammers. Quite a lot of this came from anti-virus products, and it was pointed that the vendors were often breaking the antispam laws by sending messages saying 'Product X found worm Y in the message you sent'. If worm Y was known to use address forgery, the only conceivable purpose of sending such a message to the party who hadn't sent the offending message was to advertise product X. At the same time, there's been a huge increase in mule recruitment spam.

21.2.2.5 DNS Security and Pharming

I've given two examples so far of attacks in which a user is duped by being directed to a malicious DNS server, with the result that when he tries to go to his bank website he ends up entering his password into a fake one instead. This is generally referred to as *pharming*. I mentioned drive-by pharming, in which people's home routers are reconfigured by malicious javascript in web pages they download, and rogue access points in which the attacker offers a WiFi service to the victim and then has complete control over all his unprotected traffic.

There are a number of other variants on this theme, including feeding false DNS records to genuine servers. Older DNS servers would accept additional records without checking; if they asked your server where X was, you could volunteer an IP address for Y as well. This has been fixed but there are still older servers in use that are vulnerable. Such attacks are often referred to as *DNS cache poisoning* as they basically affect users who trust the information about the target that's cached by the attacked machine. They've been used not just for pharming but also for simple vandalism, such as replacing the web site of a target company with something offensive. They can also be used for censorship; China has used DNS spoofing against dissident websites for years, and by 2007 was also using it to make Voice of America news unavailable [1297].

A number of researchers have worked on a proposed upgrade to the security of DNS, but they have turned out to be hard to deploy for economic reasons; most of the things that secure DNS would do can be done by TLS without the need for new infrastructure, and individual network operators don't get enough benefit from DNS security until enough other operators have adopted them first [997].

21.3 Trojans, Viruses, Worms and Rootkits

Computer security experts have long been aware of the threat from malicious code. The first such programs were *Trojan Horses*, named after the horse the Greeks left supposedly as a gift for the Trojans but which contained soldiers who opened the gates of Troy to the Greek army. The use of the term for malicious code goes back many years (see the discussion in [774] p 7). There are also *viruses* and *worms*, which are self-propagating malicious programs, and to which I've referred in earlier chapters. There is debate about their precise definitions; the common usage is that a Trojan is a program that does something malicious (such as capturing passwords) when run by an unsuspecting user, while a worm is something that replicates and a virus is a worm which replicates by attaching itself to other programs.

Finally, the most rapidly-growing problem is the *rootkit* — a piece of software that once installed on a machine surreptitiously places it under remote control. Rootkits can be used for targeted attacks (law enforcement agencies use them to turn suspects' laptops into listening devices) or for financial fraud (they may come with keyloggers that capture passwords). One of the most salient features of rootkits nowadays is stealth; they try to hide from the operating system so that they can't be located and removed using standard tools. But sooner or later rootkits are identified and tools to remove them are written. On the other side of the coin, most PCs infected in this way end up in botnets, so another way of framing the problem is how botnets are set up, maintained and used. The rootkit vendors now do after-sales-service, supplying their customers the botnet herders with the tools to upgrade rootkits for which removal tools are becoming available.

21.3.1 Early History of Malicious Code

Malicious code, or *malware*, seems likely to appear whenever a large enough number of users share a computing platform. It goes back at least to the early 1960's, when machines were slow and their CPU cycles were carefully rationed between different groups of users — with students often at the tail of the queue. Students invented tricks such as writing computer games with a Trojan inside to check if the program is running as root, and if so to create an extra privileged account with a known password. By the 1970s, large time-sharing systems at universities were the target of more and more pranks involving Trojans. All sorts of tricks were developed. In 1978, John Shoch and Jon Hupp of Xerox PARC wrote a program they called a *worm*, which replicated itself across a network looking for idle processors so it could assign them tasks. They discussed this in a paper in 1982 [1164].

In 1984, Ken Thompson wrote a classic paper 'On Trusting Trust', in which he showed that even if the source code for a system were carefully

inspected and known to be free of vulnerabilities, a trapdoor could still be inserted [1247]. Thompson's trick was to build the trapdoor into the compiler. If this recognized that it was compiling the login program, it would insert a trapdoor such as a master password that would work on any account. (This developed an idea first floated by Paul Karger and Robert Schell during the Multics evaluation in 1974 [693].) Of course, someone might try to stop this by examining the source code for the compiler, and then compiling it again from scratch. So the next step is to see to it that, if the compiler recognizes that it's compiling itself, it inserts the vulnerability even if it's not present in the source. So even if you can buy a system with verifiably secure software for the operating system, applications and tools, the compiler binary can still contain a Trojan. The moral is that vulnerabilities can be inserted at any point in the tool chain, so you can't trust a system you didn't build completely yourself.

1984 was also the year when computer viruses appeared in public following the thesis work of Fred Cohen. He performed a series of experiments with different operating systems in which he showed how code could propagate itself from one machine to another, and (as I mentioned in Chapter 8) from one compartment of a multilevel system to another. This caused alarm and consternation, and within about three years we started to see the first real live viruses in the wild[1]. Almost all of them were PC viruses as DOS was the predominant operating system. They spread from one user to another when users shared programs on diskettes or via bulletin boards.

One early innovation was the 'Christma' virus, which spread round IBM mainframes in December 1987. It was a program written in the mainframe command language REXX that had a header saying 'Don't read me, EXEC me' and code that, if executed, drew a Christmas tree on the screen — then sent itself to everyone in the user's contacts file. It was written as a prank, rather than out of malice; and by using the network (IBM's BITNET) to spread, it was ahead of its time.

The next year came the Internet worm, which alerted the press and the general public to the problem.

21.3.2 The Internet Worm

The first famous case of a service denial-attack was the Internet worm of November 1988 [421]. This was a program written by Robert Morris Jr that exploited a number of vulnerabilities to spread from one machine to another. Some of these were general (e.g. 432 common passwords were used in a guessing attack, and opportunistic use was made of `.rhosts` files), and others

[1]That's when I first came across them, as a security guy working in a bank; we now learn that the first ever computer virus in the wild was written for the Apple II by a 9th-grader in 1981 [1101].

were system specific (problems with `sendmail`, and the `fingerd` bug mentioned in section 4.4.1). The worm took steps to camouflage itself; it was called `sh` and it encrypted its data strings (albeit with a Caesar cipher).

Its author claimed that this code was not a deliberate attack on the Internet — merely an experiment to see whether code could replicate from one machine to another. It was successful. It also had a bug. It should have recognised already infected machines, and not infected them again, but this feature didn't work. The result was a huge volume of communications traffic that completely clogged up the Internet.

Given that the Internet (or more accurately, its predecessor the Arpanet) had been designed to provide a very high degree of resilience against attacks — up to and including a strategic nuclear strike — it was remarkable that a program written by a student could disable it completely.

What's less often remarked on is that the mess was cleaned up and normal service restored within a day or two; that it only affected Berkeley Unix and its derivatives (which may say something about the dangers of the Microsoft monoculture today); and that sites that kept their nerve and didn't pull their network connection recovered more quickly as they could find out what was happening and get the fixes.

21.3.3 How Viruses and Worms Work

A virus or worm typically has two components — a replication mechanism and a payload. A worm simply makes a copy of itself somewhere else when it's run, perhaps by breaking into another system (as the Internet worm did) or mailing itself as an attachment to the addresses on the infected system's address list (as many recent worms have done). In the days of DOS viruses, the commonest way for a virus to replicate was to append itself to an executable file and patch itself in, so that the execution path jumps to the virus code and then back to the original program.

Given a specific platform, there are usually additional tricks available to the virus writer. For example, if the target system was a DOS PC with a file called ACCOUNTS.EXE, one could introduce a file called ACCOUNTS.COM, which DOS will execute in preference. DOS viruses could also attack the boot sector or the partition table, and there are even printable viruses — viruses all of whose opcodes are printable ASCII characters, so that they can even propagate on paper. A number of DOS viruses are examined in detail in [817].

The second component of a virus is the payload. This will usually be activated by a trigger, such as a date, and may then do one or more of a number of bad things:

- make selective or random changes to the machine's protection state (this is what we worried about with multilevel secure systems);

- make changes to user data (some early viruses would trash your hard disk while some recent ones encrypt your disk and ask you to pay a ransom for the decryption key);
- lock the network (e.g., start replicating at maximum speed);
- perform some nefarious task (e.g. use the CPU for DES keysearch);
- get your modem to phone a premium-rate number in order to transfer money from you to a telephone scamster;
- install spyware or adware in your machine. This might just tell marketers what you do online — but it might steal your bank passwords and extract money from your account;
- install a *rootkit* — software that hides in your machine having taken it over. This is typically used to recruit your machine into a botnet, so that it can be used later for spam, phishing and distributed denial of service attacks at the botnet herder's pleasure.

The history of malware, and of countermeasures, has some interesting twists and turns.

21.3.4 The History of Malware

By the late 1980s and early 1990s, PC viruses had become such a problem that they gave rise to a whole industry of anti-virus software writers and consultants. Many people thought that this couldn't last, and that the move from DOS to 'proper' operating systems such as Windows would solve the problem. Some of the anti-virus pioneers even sold their companies; one of them tells his story in [1198].

However, the move to 32-bit operating systems gave only temporary respite. Soon, the spread of interpreted languages provided fertile soil for mischief. Bad Java applets flourished in the late 1990s as people found ways of penetrating Java implementations in browsers [859]. By the start of the 21st century, the main vector was the macro languages in products such as Word, and the main transmission mechanism had become the Internet [95, 209]; by 2000, macro viruses accounted for almost all incidents of mobile malicious code. Indeed, an insider says that the net 'saved' the antivirus industry [669]. A more cynical view is that the industry was never really under threat, as people will always want to share code and data, and in the absence of trustworthy computing platforms one can expect malware to exploit whichever sharing mechanisms they use. Another view is that Microsoft is responsible as they were reckless in incorporating such powerful scripting capabilities in all sorts of products. As they say, your mileage may vary.

In passing, it's worth noting that malicious data can also be a problem. An interesting example is related by David Mazières and Frans Kaashoek who

operated an anonymous remailer at MIT. This device decrypted incoming messages from anywhere on the net, uncompressed them and acted on them. Someone sent them a series of 25 Mbyte messages consisting of a single line of text repeated over and over; these compressed very well and so were only small ciphertexts when input, but when uncompressed they quickly filled up the spool file and crashed the system [849]. There are similar attacks on other programs that do decompression such as MPEG decoders. However, the most egregious cases involve not malicious data but malicious code.

Anyway, the next phase of malware evolution may have been the 'Love Bug' virus in 2000. This was actually a self-propagating worm; it propagated by sending itself to everyone in the victim's address book, and the subject line 'I love you' was calculated to get people to open it. In theory, companies can defend themselves against such things by filtering out Microsoft executables; in practice, life isn't so simple. A large Canadian company with 85,000 staff did just this, but many of their staff had personal accounts at web-based email services, and so the Love Bug virus got into the company without going through the mail filter at the firewall. The company had configured its employees' mail clients so that each of them had the entire corporate directory in her personal address book. The result was meltdown as 85,000 mail clients each tried to send an email to each of 85,000 addresses. The Love Bug was followed by a number of similar worms, which persuaded people to click on them by offering pictures of celebs such as Anna Kournikova, Britney Spears and Paris Hilton. There were also 'flash worms' that propagated by scanning the whole Internet for machines that were vulnerable to some exploit or other, and taking them over; worms of this type, such as Code Red and Slammer, infected all vulnerable machines within hours or even minutes, and caused some alarm about what sort of defences might possibly react in time [1220].

At about the same time, in the early 2000s, we saw a significant rise in the amount of *spyware* and *adware*. Spyware is technology that collects and forwards information about computer use without the owner's authorization, or with at best a a popup box that asks users to agree to perform some obscure function, so that even those who don't just reflexively click it away will not really know what they're agreeing to. This doesn't pass muster as 'consent' under European data-protection and unfair-contracts laws, but enforcement is weak. Adware may bombard the user with advertising popups and can be bundled with spyware. The vendors of this tiresome crud have even sued antivirus companies who blacklisted their wares. This all complicates everything.

A large change came about in 2004 or so. Until then, we saw a huge range of different viruses and payloads. Most virus writers did so for fun, for bragging rights, to impress their girlfriends — basically, they were amateurs. Since then, the emergence of an organised criminal economy in information goods has made the whole business much more professional. The goal of the malware

writers is to recruit machines that can be sold on for cash to botnet herders and for other exploits.

Most viruses in the 1980s and 1990s were very flaky; although tens of thousands of different viruses were reported, very few actually spread in the wild. It's actually rather difficult to write a virus that spreads properly; if it's not infectious enough it won't spread, while if you make it too infectious then it gets noticed quickly and dealt with. However, those viruses that did spread often spread very widely and infected millions of machines.

A widespread self-replicating worm may bring a huge ego boost to a teenage programmer, but for the Mafia it's not optimal. Such a worm becomes headline news and within a few hours the world's anti-virus vendors are upgrading their products to detect and remove it. Even the mass media get in on the act, telling the public not to click on any link in such-and-such an email. Now that the writers are focussed on money rather than bragging rights, they release more attacks but limited ones. Furthermore, rather than using self-replicating worms — which attract attention by clogging up the Internet — the modern trend is towards manually-controlled exploit campaigns.

In September 2007 the largest botnet was perhaps the Storm network, with around a million machines. Its herders are constantly recruiting more machines to it, using one theme after another. For example, following the start of the National Football League season on September 6th, they sent out spam on September 9th saying simply 'Football . . . Need we say more? Know all the games, what time, what channel and all the stats. Never be in the dark again with this online game tracker', following by a link to a URL from which the gullible download a file called tracker.exe that installs a rootkit in their machine. Using techniques like this — essentially, professional online marketing — they constantly grow their network. And although the media refer to Storm as a 'worm', it isn't really: it's a Trojan and a rootkit. Victims have to click away several warnings before they install it; Windows warns them that it isn't signed and asks them if they really want to install it. However, Windows pops up so many annoying dialog boxes that most people are well trained to click them away. In the case of Storm, it was targeted by Microsoft's malicious software removal tool on September 11th, and Redmond reported that over a quarter of a million machines had been cleaned; they also estimated that Storm had half a million active machines, with perhaps a few hundred thousand that were not being actively used. The network — the most powerful supercomputer on the planet — earned its living by being rented out to pump-and-dump operators and pharmacy scammers [742]. Two other networks were also identified as having over half a million bots; Gozi and Nugache use the same peer-to-peer architecture as Storm, and by the end of 2007 these networks were getting increasingly sophisticated and exploring new criminal business models [1134].

So the malware business now operates on an industrial scale, with the top botnet herders controlling roughly the same number of machines as Google. Big business has been built on the fact that users have been trained to click on stuff. As malware goes industrial, Trojans are becoming more common than viruses; when the latter email themselves out from an infected machine, they draw attention to themselves and the machine's more likely to get cleaned up, while with Trojans the botnet herder sends the infectious traffic directly, which also given him better control [1239]. And once you install something, there's no telling whether it's a rootkit, or malicious spyware that will use a keystroke logger to steal your banking passwords, or a 'normal' piece of spyware that will simply collect your personal data for sale to the highest bidder. Truth to tell, the categories are hard to separate cleanly.

21.3.5 Countermeasures

Within a few months of the first PC viruses appearing in the wild in 1987, companies had set up to sell antivirus software. This led to an arms race in which each tried to outwit the other. Early software came in basically two flavours — *scanners* and *checksummers*.

Scanners are programs that search executable files for a string of bytes known to be from an identified virus. Virus writers responded in various ways, such as specific counterattacks on popular antivirus programs; the most general technique is *polymorphism*. The idea here is to change the code each time the virus or worm replicates, to make it harder to write effective scanners. The usual technique is to encrypt the code using a simple cipher, and have a small header that contains decryption code. With each replication, the virus re-encrypts itself under a different key, and tweaks the decryption code by substituting equivalent sequences of instructions.

Checksummers keep a list of all the authorised executables on the system, together with checksums of the original versions, typically computed using a hash function. The main countermeasure is *stealth*, which in this context means that the virus watches out for operating system calls of the kind used by the checksummer and hides itself whenever a check is being done.

Researchers have also looked into the theory of malware replication. In order for a virus infestation to be self-sustaining, it needs to pass an *epidemic threshold* — at which its rate of replication exceeds the rate at which it's removed [711]. This depends not just on the infectivity of the virus itself but on the number (and proportion) of connected machines that are vulnerable. Epidemic models from medicine go over to some extent, though they are limited by the different topology of software intercourse (sharing of software is highly localised) and so predict higher infection rates than are actually observed. (I'll return to topology later.) People have also tried to use immune-system models to develop distributed strategies for malware detection [482].

One medical lesson which does seem to apply is that the most effective organisational countermeasure is centralised reporting and response using selective vaccination [712].

In the practical world, antivirus software and managerial discipline are to a certain extent substitutes, but to be really effective, you have to combine tools, incentives and management. In the old days of DOS-based file viruses, this came down to providing a central reporting point for all incidents, and controlling all software loaded on the organisation's machines. The main risks were files coming in via PCs used at home both for work and for other things (such as kids playing games), and files coming in from other organisations. But how do you get staff to sweep all incoming email and diskettes for viruses? One effective strategy, adopted at a London law firm, was to reward whoever found a virus with a box of chocolates — which would then be invoiced to the company that had sent the infected file.

Now that malware arrives mostly in email attachments or in web pages, things are often more technical, with automatic screening and central reporting. A company may filter executables out at the firewall, and see to it that users have prudent default settings on their systems — such as disabling active content on browsers and macros in word processing documents. Of course, this creates a clash with usability. People will also create all sorts of unauthorized communications channels, so you have to assume that screening can't be perfect; staff must still be trained not to open suspicious email attachments, and in recovery procedures so they can deal with infected backups. In short, the issues are more complex and diffuse. But as with the organic kind of disease, prevention is better than cure; and software hygiene can be integrated with controls on illegal software copying and unauthorised private use of equipment.

Recently, antivirus software seems to be getting steadily less effective. The commercialisation of botnets and of machine exploitation has meant that malware writers have decent tools and training. Almost all Trojans and other exploits are undetectable by the current antivirus products when first launched — as their writers test them properly — and many of them run their course (by recruiting their target number of machines) without coming to the attention of the antivirus industry. The net effect is that while antivirus software might have detected almost all of the exploits in circulation in the early 2000s, by 2007 the typical product might detect only a third of them.

And as for the rootkits that the exploits leave behind, they are also much better written than a few years ago, and rarely cause trouble for the owner of the machine on which they're installed. Some rootkits even install up-to-date antivirus software to stop any competing botnet from taking the machine over. They also use all sorts of stealth techniques to hide from detectors. What's more, the specialists who sell the rootkits provide after-sales service; if a removal kit is shipped, the rootkit vendor will rapidly ship countermeasures.

It's not at all clear that technical defences are keeping up with malware. On the global scale, police action against the large gangs is needed, and although it's starting to ramp up, there's a long way to go. Well-run firms can use managerial discipline to contain the threat, but for private users of Windows machines, the outlook isn't particularly rosy. One survey suggested that 8% of sales of new PCs are to people who've simply given up on machines that have become so infested with adware and other crud as to become unusable [325]; and there is a growing threat from *keyloggers* that capture everything the user does at his machine. Some of these are simply spyware that sells information to marketers; others look out for bank passwords and other key data that can be used to commit fraud directly.

21.4 Defense Against Network Attack

In defending against network attack, there are broadly speaking four sets of available tools.

1. First is management — keeping your systems up-to-date and configured in ways that will minimise the attack surface;
2. Next is filtering — the use of firewalls to stop bad things like Trojans and network exploits, and to detect signs of attack and compromise if anything gets through;
3. Next is intrusion detection — having programs monitoring your networks and machines for signs of malicious behaviour;
4. Finally there's encryption — protocols such as TLS and SSH that enable you to protect specific parts of the network against particular attacks.

Let's work through these in turn.

21.4.1 Configuration Management and Operational Security

The great majority of technical attacks on systems in the period 2000–07 exploited already known vulnerabilities. The typical cycle is that Microsoft announces a set of security patches once a month; as soon as they come out, the attackers start reverse engineering them; within a few days, the vulnerabilities that they fixed are understood and exploits appear. A well-run firm will test its operational systems quickly on Patch Tuesday and apply the patches, provided they don't break anything important. If they do break something, that will be fixed as quickly as reasonably possible.

Tight configuration management is not just about patches, though. Many software products ship with unsafe defaults, such as well-known default

passwords. It's a good idea to have someone whose job it is to understand and deal with such problems. It's also common to remove unnecessary services from machines; there is usually no reason for every workstation in your company to be running a mail server, and ftp server and DNS, and stripping things down can greatly reduce the attack surface. Frequent reinstallation is another powerful tool: when this was first tried at MIT during Project Athena, a policy of overnight reinstallation of all software greatly cut the number of sysadmins needed to look after student machines. Operations like call centres often do the same; that way if anyone wants to install unauthorised software they have to do it again every shift, and are more likely to get caught. There are also network configuration issues: you want to know your network's topology, and have some means of hunting down things like rogue access points. If all this is done competently, then you can deal with most of the common technical attacks. (You'll need separate procedures to deal with bugs that arise in your own code, but as most software is bought rather than written these days, configuration management is most of the battle.)

There are many tools to help the sysadmin in these tasks. Some enable you to do centralized version control so that patches can be applied overnight and everything kept in synch; others look for vulnerabilities in your network. When the first such tool came out (Satan [503]) there was quite a lot of controversy; this has led to some countries passing laws against 'hacking tools'. Now there are dozens of such tools, but they have to be used with care.

However, a strategy of having your system administrators stop all vulnerabilities at source is harder than it looks; even diligent organisations may find it's just too expensive to fix all the security holes at once. Patches may break critical applications, and it seems to be a general rule that an organisation's most critical systems run on the least secure machines, as administrators have not dared to apply upgrades and patches for fear of losing service.

This leads us to operational security, and the use of filtering tools such as firewalls.

Operational security, as mentioned in Chapter 2 and Chapter 8, is about training staff to not expose systems by foolish actions. There we were largely interested in social engineering attacks involving the telephone; the main way of getting unauthorised access to information is still to phone up and pretend to be someone who's entitled to know.

Now the main way machines get compromised in 2007 is because people click on links in email that cause them to download and install rootkits. Of course you must train your staff to not click on links in mail, but don't expect that this alone will fix the problem; many banks and other businesses expect their customers to click on links, and many of your staff will have to do some clicking to get their work done. You can shield low-grade staff by not giving them administrator access to their machines, and you can shield your creative

staff by letting them buy Macs; but there is still going to be a residual risk. One common way of dealing with it is to strip out all executables at your firewall.

21.4.2 Filtering: Firewalls, Spam Filters, Censorware and Wiretaps

The most widely sold solution to the 'problems of Internet security' is the *firewall*. This is a machine which stands between a local system and the Internet and filters out traffic that might be harmful. The idea of a 'solution in a box' has great appeal to many organisations, and is now so widely accepted that it's seen as an essential part of corporate due diligence. (This in itself creates a risk — many firms prefer expensive firewalls to good ones.)

Firewalls are just one example of systems that examine streams of packets and perform filtering operations. Bad packets may be thrown away, or modified in such a way as to make them harmless. They may also be copied to a log or audit trail. Very similar systems are also used for Internet censorship and for law-enforcement wiretapping; almost everything I'll discuss in this section goes across to those applications too. Developments in any of these fields potentially affect the others; and actual systems may have overlapping functions. For example, many corporate firewalls or mail filters screen out pornography, and some even block bad language, while ISP systems that censor child pornography or dissenting political speech may report the perpetrators automatically to the authorities.

Filters come in basically three flavours, depending on whether they operate at the IP packet level, at the TCP session level or at the application level.

21.4.2.1 Packet Filtering

The simplest kind of filter merely inspects packet addresses and port numbers. This functionality is also available in routers, in Linux and indeed in Windows. A firewall can block IP spoofing by ensuring that only 'local' packets leave a network, and only 'foreign' ones enter. It can also stop denial-of-service attacks in which malformed packets are sent to a host. It's also easy to block traffic to or from 'known bad' IP addresses. For example, IP filtering is a major component of the censorship mechanisms in the Great Firewall of China; a list of bad IP addresses can be kept in router hardware, which enables packet filtering to be done at great speed.

Basic packet filtering is also available as standard on most machines and can be used for more mundane firewalling tasks. For example, packet filters can be configured to block all traffic except that arriving on specific port numbers. The configuration might be initially to allow the ports used by common services such as email and web traffic, and then open up ports as the protected machine or subnet uses them.

However, packet filters can be defeated by a number of tricks. For example, a packet can be fragmented in such a way that the initial fragment (which passes the firewall's inspection) is overwritten by a subsequent fragment, thereby replacing the source address with one that violates the firewall's security policy. Another limitation is that maintaining a blacklist is difficult, and especially so when it's not the IP address specifically you want to block, but something that resolves into an IP address, especially on a transient basis. For example, the phishermen are starting to use tricks like fast-flux in which a site's IP address changes several times an hour.

21.4.2.2 Circuit Gateways

The next step up is a more complex arrangement, a *circuit gateway*, that operates at level 4, typically by reassembling and examining all the packets in each TCP session. This is more expensive than simple packet filtering; its main advantage is that it can also provide the added functionality of a *virtual private network* whereby corporate traffic passed over the Internet is encrypted from firewall to firewall. I'll discuss the IPSEC protocol that's used for this in the last section of this chapter.

TCP-level filtering can be used to do a few more things, such as DNS filtering. However, it can't screen out bad things at the application level, such as malicious code, image spam and unlawful images of child abuse. Thus it may often be programmed to direct certain types of traffic to specific application filters. An example is British Telecom's CleanFeed system, which tries to prevent its customers getting access to child pornography. As some bad sites are hosted on public web services and blocking all the web pages at the service would be excessive, TCP/IP filtering is used to redirect traffic with such sites to a proxy that can examine it in detail.

21.4.2.3 Application Relays

The third type of firewall is the *application relay*, which acts as a proxy for one or more services. Examples are mail filters that try to weed out spam, and web proxies that block or remove undesirable content. The classic example is a corporate rule about stripping out code, be it straightforward executables, active content in web pages, macros from incoming Word documents. Over the period 2000–07, this has been a constant arms race between the firewall vendors, the spammers, and people trying to circumvent controls to get their work done.

The flood of Word macro viruses around 2000 led many firms to strip out Word macros (or even all Word documents) from email. Workers got round this by zipping documents first. Firewalls started unzipping them to inspect them, whereupon people started encrypting them using zip's password feature, and

putting the password in the email plaintext. Once firewalls started to cope with this, the spammers started putting zip passwords in images attached to the mail along with the zip file. Eventually, many companies started adopting a policy of not sending out Word documents, but Pdf documents instead; this not only made it easier to get past firewalls, but also stopped people carelessly sending out documents containing the last few dozen edits. Needless to say, the spammers now send out Pdf attachments — and their botnets have the power to make all the attachments different, for example by combining text, and image, and a number of random color blocks for background. Rootkit executables are now often distributed as web links; August 2007 saw floods of messages telling people they'd got a card, while in September it was links to a bogus NFL site. For complete protection, you have to filter executables in your web proxy too (but this would really get in the way of users who wish to run the latest applications). There is no sign of this arms race abating.

An application relay can also turn out to be a serious bottleneck. This applies not just to the corporate application, but in censorship. An example is the Great Firewall of China, which tries to block mail and web content that refers to banned subjects. Although the firewall can block 'known bad' sites by simple IP filtering, finding forbidden words involves deep packet inspection — which needs much more horsepower. An investigation by Richard Clayton, Steven Murdoch and Robert Watson showed bad content wasn't in fact blocked; machines in China simply sent 'reset' packets to both ends of a connection on which a bad word had appeared. This was almost certainly because they needed a number of extra machines for the filtering, rather than doing it in the router; one side-effect was that you could defeat the firewall by ignoring these reset packets [308]. (Of course, someone within China who did that might eventually get a visit from the authorities.)

At the application level in particular, the pace of innovation leaves the firewall vendors (and the censors and the wiretappers) trailing behind. A good example is the move to edge-based computing. Google's word processor — Google Documents — is used by many people to edit documents online, simply to save them the cost of buying Microsoft Word. As a side-effect, its users can instantly share documents with each other, creating a new communications channel of which classical filters are unaware. So the service might be used to smuggle confidential documents out of a company, to defeat political censors, or to communicate covertly. (It even blurs the distinction between traffic and content, which is central to the legal regulation of wiretapping in most countries.) Even more esoteric communications channels are readily available — conspirators could join an online multi-user game, and pass their messages via the silver dragon in the sixth dungeon.

Another problem is that application-level filtering can be very expensive, especially of high-bandwidth web content. That's why a number of web filtering systems are hybrids, such as the CleanFeed mechanism mentioned above where only those domains that contain at least some objectionable

content are sent to full http proxy filtering. Application proxies can also interact with other protection mechanisms. Not only can spammers (and others) use encryption to defeat content inspection; but some corporate web proxies are set up to break encryption by doing middleperson attacks on TLS. Even if you think you're giving an encrypted credit card number to Amazon, your encrypted session may just be with your employer's web proxy, while it runs another encrypted session with Amazon's web server. I'll discuss TLS later in this chapter.

21.4.2.4 Ingress Versus Egress Filtering

At present, most firewalls look outwards and try to keep bad things out, but a growing number look inwards and try to stop bad things leaving. The pioneers were military mail systems that monitor outgoing traffic to ensure that nothing classified goes out in the clear; around 2005 some ISPs started looking at outgoing mail traffic to try to detect spam. The reason is that ISPs which host lots of infected machines and thus pump out lots of spam damage their peering relationships with other ISPs, which costs real money; so various systems have been developed to help them spot infected machines, that can then be restricted to a 'walled garden' from which they can access anti-virus software but not much else [300].

If companies whose machines get used in service denial attacks start getting sued, as has been proposed in [1285], then egress filtering can at least in principle be used to detect and stop such attacks. However, at present the incentives just aren't there, and so although people care about spam floods, almost nobody at the ISP level bothers about packet floods. This might of course change as attacks get worse or if the regulatory environment changes.

Another possible development is egress filtering for privacy, given the rising tide of spyware. Software that 'phones home', whether for copyright enforcement and marketing purposes, can disclose highly sensitive material such as local hard disk directories. Prudent organizations will increasingly wish to monitor and control this kind of traffic. In the long term we expect that 'pervasive computing' will fill our homes with all sorts of gadgets that communicate, so I wouldn't be surprised to see home firewalls that enable the householder to control which of them 'phone home', and for what purpose.

21.4.2.5 Architecture

Many firms just buy a firewall because it's on the tick-list of due-diligence things their auditors want to see. In that case, the sensible choice is a simple filtering router, which won't need much maintanence and won't get in the way. Where security's taken seriously, one possible approach is to invest in a really serious firewall system, which might consist of a packet filter connecting the outside world to a screened subnet, also known as a *demilitarized zone*

(DMZ), which in turn contains a number of application servers or proxies to filter mail, web and other services. The DMZ may then be connected to the internal network via a further filter that does network address translation.

In [323], there is a case study of how a firewall was deployed at Hanscom Air Force Base. The work involved surveying the user community to find what network services were needed; devising a network security policy; using network monitors to discover unexpected services that were in use; and lab testing prior to installation. Once it was up and running, the problems included ongoing maintenance (due to personnel turnover), the presence of (unmonitored) communications to other military bases, and the presence of modem pools. Few non-military organizations are likely to take this much care.

An alternative approach is to have more networks, but smaller ones. At our university, we have firewalls to separate departments, although we've got a shared network backbone and there are some shared central services. There's no reason why the students and the finance department should be on the same network, and a computer science department has got quite different requirements (and users) from a department of theology — so the network security policies should be different too. In any case keeping each network small limits the scope of any compromise.

You may even find both a big corporate firewall and departmental boundaries. At defense contractors, you may expect to find not just a fancy firewall at the perimeter, but also pumps separating networks operating at different clearance levels, with filters to ensure that classified information doesn't escape either outwards or downwards (Figure 21.2).

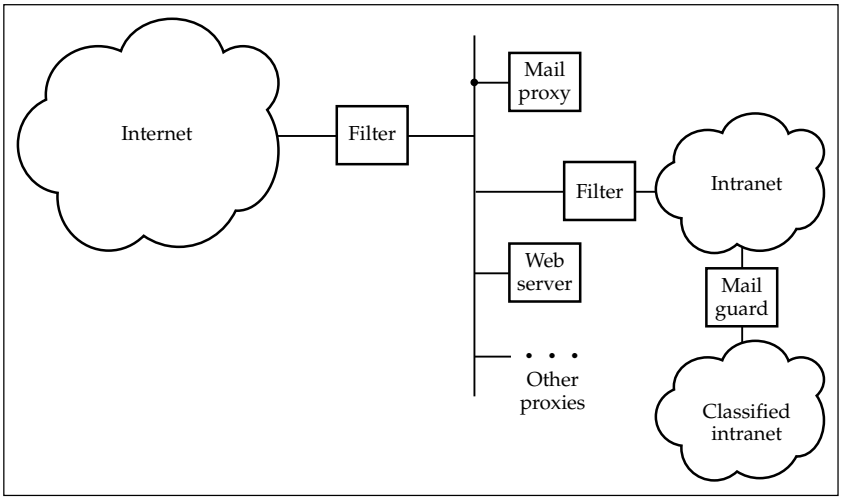

Figure 21.2: Multiple firewalls

Which is better? Well, it depends. Factors to consider when designing a network security architecture are simplicity, usability, maintainability, deperimeterisation, underblocking versus overblocking, and incentives.

First, since firewalls do only a small number of things, it's possible to make them very simple and remove many of the complex components from the underlying operating system, removing a lot of sources of vulnerability and error. If your organization has a large heterogeneous population of machines, then loading as much of the security management task as possible on a small number of simple boxes makes sense. On the other hand, if you're running something like a call centre, with a thousand identically-configured PCs, it makes sense to put your effort into keeping this configuration tight.

Second, elaborate central installations not only impose greater operational costs, but can get in the way so much that people install back doors, such as cable modems that bypass your firewall, to get their work done. In the 1990s, UK diplomats got fed up waiting for a 'secure' email system from government suppliers; they needed access to email so they could talk to people who preferred to use it. Some of them simply bought PCs from local stores and got accounts on AOL, thus exposing sensitive data to anyone who tapped the network or indeed guessed an AOL password. In fact, the diplomats of over a hundred countries had webmail accounts compromised when they were foolish enough to rely on Tor for message confidentiality, and got attacked by a malicious Tor exit node (an incident I'll discuss in section 23.4.2.) So a prudent system administrator will ensure that he knows the actual network configuration rather than just the one stated by 'policy'.

Third, firewalls (like other filtering products) tend only to work for a while until people find ways round them. Early firewalls tended to let only mail and web traffic through; so writers of applications from computer games to anonymity proxies redesigned their protocols to make the client-server traffic look as much like normal web traffic as possible. Now, of course, in the world of Web 2.0, more and more applications are actually web-based; so we can expect the same games to be played out again in the web proxy. There are particular issues with software products that insist on calling home. For example, the first time you use Windows Media Player, it tells you you need a 'security upgrade'. What's actually happening is that it 'individualizes' itself by generating a public-private keypair and sending the public key to Microsoft. If your firewall doesn't allow this, then WMP won't play protected content. Microsoft suggests you use their ISA firewall product, which will pass WMP traffic automatically. Quite a few issues of trust, transparency and competition may be raised by this!

Next, there's *deperimiterization* — the latest buzzword. Progress is making it steadily harder to put all the protection at the perimeter. The very technical ability to maintain a perimeter is undermined by the proliferation of memory sticks, of laptops, of PDAs being used for functions that used to be done on

desktop computers, and by changing business methods that involve more outsourcing of functions — whether formally to subcontractors or informally to advertising-supported web apps. If some parts of your organisation can't be controlled well (e.g. the sales force and the R&D lab) while others must be (the finance office) then separate networks are needed. The crumbling of the perimeter will be made even worse by mobility, and by the proliferation of web applications. This is complemented by a blunting of the incentive to do things at the perimeter, as useful things become harder to do. The difference between code and data is steadily eroded by new scripting languages; a determination to not allow javascript in the firm is quickly eroded by popular web sites that require it; and so on.

And then there's our old friend the Receiver Operating Characteristic or ROC curve. No filtering mechanism has complete precision, so there's inevitably a trade-off between underblocking and overblocking. If you're running a censorship system to stop kids accessing pornography in public libraries, do you underblock, and annoy parents and churches when some pictures get through, or do you overblock and get sued for infringing free-speech rights? Things are made worse by the fact that the firewall systems used to filter web content for sex, violence and bad language also tend to block free-speech sites (as many of these criticise the firewall vendors — and some offer technical advice on how to circumvent blocking.)

Finally, security depends at least as much on incentives as on technology. A sysadmin who's looking after a departmental network used by a hundred people he knows, and who will personally have to clear up any mess caused by an intrusion or a configuration error, is much more motivated than someone who's merely one member of a large team looking after thousands of machines.

21.4.3 Intrusion Detection

It's a good idea to assume that attacks will happen, and it's often cheaper to prevent some attacks and detect the rest than it is to try to prevent everything. The systems used to detect bad things happening are referred to generically as *intrusion detection systems*. The antivirus software products I discussed earlier are one example; but the term is most usually applied to boxes that sit on your network and look for signs of an attack in progress or a compromised machine [1100]. Examples include:

- spam coming from a machine in your network;
- packets with forged source addresses — such as packets that claim to be from outside a subnet coming from it, or packets that claim to be from inside arriving at it;
- a machine trying to contact a 'known bad' service such as an IRC channel that's being used to control a botnet.

In cases like this, the IDS essentially tells the sysadmin that a particular machine needs to be scrubbed and have its software reinstalled.

Other examples of intrusion detection, that we've seen in earlier chapters, are the mechanisms for detecting mobile phone cloning and fraud by bank tellers. There are also bank systems that look at customer complaints of credit card fraud to try to figure out which merchants have been leaking card data, and stock market systems that try to detect insider trading by looking for increases in trading volume prior to a price-sensitive announcement and other suspicious patterns of activity. And there are 'suspect' lists kept by airport screeners; if your name is down there, you'll be selected 'at random' for extra screening. Although these intrusion detection systems are all performing very similar tasks, their developers don't talk to each other much. One sees the same old wheels being re-invented again and again. But it's starting slowly to become a more coherent discipline, as the U.S. government has thrown hundreds of millions at the problem.

The research program actually started in the mid-1990s and was prompted by the realisation that many systems make no effective use of log and audit data. In the case of Sun's operating system Solaris, for example, we found in 1996 that the audit formats were not documented and tools to read them were not available. The audit facility seemed to have been installed to satisfy the formal checklist requirements of government systems buyers rather than to perform any useful function. There was at least the hope that improving this would help system administrators detect attacks, whether after the fact or even when they were still in progress. Since 9/11, of course, there has been a great switch of emphasis to doing data mining on large corpora of both government and commercial data, looking for conspiracies.

21.4.3.1 Types of Intrusion Detection

The simplest intrusion detection method is to sound an alarm when a threshold is passed. Three or more failed logons, a credit card expenditure of more than twice the moving average of the last three months, or a mobile phone call lasting more than six hours, might all flag the account in question for attention. More sophisticated systems generally fall into two categories.

Misuse detection systems operate using a model of the likely behaviour of an intruder. A banking system may alarm if a user draws the maximum permitted amount from a cash machine on three successive days; and a Unix intrusion detection system may look for user account takeover by alarming if a previously naive user suddenly started to use sophisticated tools like compilers. Indeed, most misuse detection systems, like antivirus scanners, look for a *signature* — a known characteristic of a particular attack.

Anomaly detection systems attempt the much harder job of looking for anomalous patterns of behaviour in the absence of a clear model of the

attacker's modus operandi. The hope is to detect attacks that have not been previously recognized and cataloged. Systems of this type often use AI techniques — neural networks have been fashionable from time to time.

The dividing line between misuse and anomaly detection is somewhat blurred. A good borderline case is Benford's law, which describes the distribution of digits in random numbers. One might expect that numbers beginning with the digits '1', '2', ... '9' would be equally common. But in fact with numbers that come from random natural sources, so that their distribution is independent of the number system in which they're expressed, the distribution is logarithmic: about 30% of decimal numbers start with '1'. Crooked clerks who think up numbers to cook the books, or even use random number generators without knowing Beford's law, are often caught using it [846]. Another borderline case is the *honey trap* — something enticing left to attract attention. I mentioned, for example, that some hospitals have dummy records with celebrities' names in order to entrap staff who don't respect medical confidentiality.

21.4.3.2 General Limitations of Intrusion Detection

Some intrusions are really obvious. If what you're worried about is a script kiddie vandalizing your corporate web site, then the obvious defence to have a machine somewhere that fetches the page regularly, inspects it, and rings a really loud alarm when it changes. (Make sure you do this via an outside proxy, and don't forget that it's not just your own systems at risk. The kiddie could replace your advertisers' pictures with porn, for example, and then you'd want to pull the links to them pretty fast.)

But in the general case, intrusion detection is hard. Cohen proved that detecting viruses (in the sense of deciding whether a program is going to do something bad) is as hard as the halting problem, so we can't ever expect a complete solution [311].

Another fundamental limitation comes from the fact that there are basically two different types of security failure — those which cause an error (which we defined in 6.3 to be an incorrect state) and those which don't. An example of the former is a theft from a bank which leaves traces on the audit trail. An example of the latter is an undetected confidentiality failure caused by a radio microphone placed by a foreign intelligence service in your room. The former can be detected (at least in principle, and forgetting for now about the halting problem) by suitable processing of the data available to you. But the latter can't be. It's a good idea to design systems so that as many failures as possible fall into the former category, but it's not always practicable [289].

There's also the matter of definitions. Some intrusion detection systems are configured to block any instances of suspicious behaviour and in extreme cases to take down the affected systems. Quite apart from opening the door to

service denial attacks, this turns the intrusion detection system into firewall or an access control mechanism; and as we've already seen, access control is in general a hard problem and incorporates all sorts of issues of security policy which people often disagree on or simply get wrong.

I prefer to define an intrusion detection system as one that monitors the logs and draws the attention of authority to suspicious occurrences. This is closer to the way mobile phone operators work. It's also critical in financial investigations; see [1095] for a discussion by a special agent with the U.S. Internal Revenue Service, of what he looks for when trying to trace hidden assets and income streams. A lot hangs on educated suspicion based on long experience. For example, a $25 utility bill may lead to a $250,000 second house hidden behind a nominee. Building an effective system means having the people, and the machines, each do the part of the job they're best at; and this means getting the machine to do the preliminary filtering.

Then there's the cost of false alarms. For example, I used to go to San Francisco every May, and I got used to the fact that after I'd used my UK debit card in an ATM five days in a row, it would stop working. This not only upsets the customer, but the villains quickly learn to exploit it. (So do the customers — I just started making sure I got enough dollars out in the first five days to last me the whole trip.) As in so many security engineering problems, the trade-off between the fraud rate and the insult rate is the critical one — and, as discussed in section 15.9, you can't expect to improve this trade-off simply by looking at lots of different indicators. In general, you must expect that an opponent will always get past the threshold if he's patient enough and either does the attack very slowly, or does a large number of small attacks.

A difficult policy problem with commercial intrusion detection systems is *redlining*. When insurance companies used claim statistics on postcodes to decide the level of premiums to charge, it was found that many poor and black areas suffered high premiums or were excluded altogether from cover. In a number of jurisdictions this is now illegal. In general, if you build an intrusion detection system based on data mining techniques, you are at serious risk of discriminating. If you use neural network techniques, you'll have no way of explaining to a court what the rules underlying your decisions are, so defending yourself could be hard. Opaque rules can also contravene European data protection law, which entitles citizens to know the algorithms used to process their personal data.

Already in 1997, systems introduced to profile U.S. airline passengers for terrorism risk, so they could be subjected to more stringent screening, were denounced by the American-Arab Anti-Discrimination Committee [823]. Since 9/11 such problems have become much worse. How do we judge the balance point beyond which we just radicalize people and breed more attacks? I'll come back to this in Part III.

21.4.4 Specific Problems Detecting Network Attacks

Turning now to the specific problem of detecting network intrusion, the problem is much harder than (say) detecting mobile phone cloning for a number of reasons. Network intrusion detection products still don't work very well, with both high missed alarm and false alarm rates. It's common not to detect actual intrusions until afterwards — although once one is detected by other means, the traces can be found on the logs.

The reasons for the poor performance include the following, in no particular order.

- The Internet is a very noisy environment — not just at the level of content but also at the packet level. A large amount of random crud arrives at any substantial site, and enough of it can be interpreted as hostile to provide a significant false alarm rate. A survey by Bellovin [149] reports that many bad packets result from software bugs; others are the fault of out-of-date or corrupt DNS data; and some are local packets that escaped, travelled the world and returned.
- There are 'too few attacks'. If there are ten real attacks per million sessions — which is almost certainly an overestimate — then even if the system has a false alarm rate as low as 0.1%, the ratio of false to real alarms will be 100. We talked about similar problems with burglar alarms; it's also a well known problem for medics running screening programs for diseases like HIV where the test error exceeds the organism's prevalence. In general, where the signal is far below the noise, the guards get tired and even the genuine alarms get missed.
- Many network attacks are specific to particular versions of software, so a general misuse detection tool must have a large and constantly changing library of attack signatures.
- In many cases, commercial organisations appear to buy intrusion detection systems simply in order to tick a 'due diligence' box to satisfy insurers or consultants. That means the products aren't always kept up to date.
- Encrypted traffic can't easily be subjected to content analysis any more than it can be filtered for malicious code.
- The issues we discussed in the context of firewalls largely apply to intrusion detection too. You can filter at the packet layer, which is fast but can be defeated by packet fragmentation; or you can reconstruct each session, which takes more computation and so is not really suitable for network backbones; or you can examine application data, which is more expensive still — and needs to be constantly updated to cope with the arrival of new applications and attacks.

- You may have to do intrusion detection both locally and globally. The antivirus side of things may have to be done on local machines, especially if the malware arrives on encrypted web sessions; on the other hand, some attacks are *stealthy* — the opponent sends 1–2 packets per day to each of maybe 100,000 hosts. Such attacks are unlikely to be found by local monitoring; you need a central monitor that keeps histograms are kept of packets by source and destination address and by port.

So it appears unlikely that a single-product solution will do the trick. Future intrusion detection systems are likely to involve the coordination of a number of monitoring mechanisms at different levels both in the network (backbone, LAN, individual machine) and in the protocol stack (packet, session and application).

21.4.5 Encryption

In the context of preventing network attacks, many people have been conditioned to think of encryption. Encryption usually does a lot less than you might hope, as the quote from Butler Lampson and Roger Needham at the head of this chapter suggests. But it can sometimes be useful. Here I'm going to describe briefly the four most relevant network encryption scenarios: SSH; the local link protection offered by WiFi, Bluetooth and HomePlug; IPSec; and TLS. Finally I'll briefly discuss public key infrastructures (PKI), which are used to support the last two of these.

21.4.5.1 SSH

When I use my laptop to read email on my desktop machine, or do anything with any other machine in our lab for that matter, I use a protocol called *secure shell* (SSH) which provides encrypted links between Unix and Windows hosts [1369, 1, 988]. So when I come in from home over the net, my traffic is protected, and when I log on from the PC at my desk to another machine in the lab, the password I use doesn't go across the LAN in the clear.

SSH was initially written in 1995 by Tatu Ylönen, a researcher at Helsinki University of Technology in Finland, following a password-sniffing attack there. It not only sets up encrypted connections between machines, so that logon passwords don't travel across the network in the clear; it also supports other useful features, such as forwarding X sessions, which led to its rapid adoption. (In fact it's a classic case study in how to get a security product accepted in the marketplace; see [1083] for an analysis. Normally people don't want to use encryption products until a lot of other people are using them too, because of network effects; so the trick is to bundle some real other benefits with the product.)

There is a proprietary SSH product from Ylönen's company, and a number of open implementations such as OpenSSH and Putty; there's also an associated file transfer protocol SCP ('secure copy'). There are various configuration options, but in the most straightforward one, each machine has a public-private keypair. The private key is protected by a passphrase that the user types at the keyboard. To connect from (say) my laptop to a server at the lab, I ensure that my public key is loaded on, and trusted by, the server. Manual key installation is at once a strength and a weakness; it's strong in that management is intuitive, and weak as it doesn't scale particularly well. In any case, when I wish to log on to the server I'm prompter for my passphrase; a key is then set up; and the traffic is both encrypted and authenticated. Fresh keys are set up after an hour, or after a Gigabyte of traffic has been exchanged.

Possible problems with the use of SSH include the fact that the earliest version, SSH 1.0, is vulnerable to middleperson attacks because of a poor key-exchange protocol; and that if you're typing at the keyboard one character at a time, then each character gets sent in its own packet. The packet inter-arrival times can leak a surprising amount of information about what you're typing [1203]. However, the worst is probably that most SSH keys are stored in the clear, without being protected by a password at all. The consequence is that if a machine is compromised, the same can happen to every other machine that trusts an SSH key installed on it.

21.4.5.2 WiFi

WiFi is a technology used for wireless local area networks, and is very widely used: people use it at home to connect PCs to a home router, and businesses use it too, connecting devices such as tills and payment terminals as well as PCs. Games consoles and even mobile phones make increasing use of wireless LANs.

Wifi has come with a series of encryption protocols since its launch in 1997. The first widely-used one, WEP (for *wired equivalent privacy*), was shown to be fairly easily broken, even when configured correctly. Standardised with IEEE 802.11 in 1999, WEP uses the RC4 stream cipher to encrypt data with only a cyclic redundancy check for integrity. Nikita Borisov, Ian Goldberg and David Wagner showed that this led to attacks in depth [210]. Known plaintext allows keystream to be stripped off and reused; in addition, the initial values used in encryption were only 24 bits, which enabled IV collisions to be found leading to further depth attacks. False messages could be encrypted and injected into a wireless LAN, opening it to other attacks. What's more, the key was only 40 bits long in early implementations, because of U.S. export rules; so keys could be brute-forced.

That merely whetted cryptanalysts' appetite. Shortly afterwards, Scott Fluhrer, Itzhak Mantin and Adi Shamir found a really devastating attack.

It is an initialisation weakness in the RC4 stream cipher that interacts with the way in which WEP set up its initialization vectors [479]; in short order, Adam Stubblefield, John Ioannidis and Avi Rubin turned this into a working attack on WEP [1230]. Vendors bodged up their products so they would not use the specific weak keys exploited in the initial attack programs; later programs used a wider range of weak keys, and the attacks steadily improved. The history of the attack evolution is told in [993]; the latest attack, by Erik Tews, Ralf-Philipp Weinmann and Andrei Pyshkin, recovers 95% of all keys within 85,000 packets [1245]. Now there are publicly-available tools that will extract WEP keys after observing a few minutes' traffic.

Stronger encryption systems, known as Wi-Fi Protected Access (WPA), aim to solve this problem and are available on most new products. WPA shipped in 2003, and was an intermediate solution that still uses RC4. WPA2 shipped in 2004; it is also called the Robust Security Network (RSN) and uses the AES block cipher in counter mode with CBC-MAC. Within a few years, as older machines become obsolete, WPA2 should solve the cipher security problem.

So what are we to make of WiFi security? There has been a lot of noise in the press about how people should set passwords on their home routers, in case a bad man stops outside your house and uses your network to download child porn. However, a straw poll of security exports at WEIS 2006 showed that most did not bother to encrypt their home networks; drive-by downloads are a fairly remote threat. For most people in the UK or America, it's just convenient to have an open network for your guests to use, and so that you and your neighbours can use each others' networks as backups. Things are different in countries where you pay for download bandwidth; there, home router passwords are mostly set.

Things are different for businesses because of the possibility of targeted attacks. If you use a Windows machine with Windows shares open, then someone on your LAN can probably use that to infect you with malware. A random home owner may not be at much risk — with botnets trading at about a dollar a machine, it's not worth someone's while to drive around town infecting machines by hand. But if you're a high-value target, than the odds change significantly. In March 2007, retail chain TJ Maxx reported that some 45.7 million credit card numbers had been stolen from its systems; these card numbers largely related to sales in 2003 and 2004, and had been stolen from 2005 but discovered only in December 2006. The Wall Street Journal reported that an insecure WiFi connection in St Paul, Mn., was to blame [1014]; the company's SEC filing about the incident is at [1252], and the Canadian Privacy Commissioner concluded that 'The company collected too much personal information, kept it too long and relied on weak encryption technology to protect it — putting the privacy of millions of its customers at risk' [1047]. Banks sued the company, with VISA claiming fraud losses of over $68m from the compromise of 65 million accounts; the banks eventually

settled for damages of $41m [544]. Since June 2007, the banking industry's Payment Card Industry Data Security Standard (PCI DSS) requires companies processing credit card data to meet certain data security standards, and VISA or Mastercard can fine member banks whose merchants don't comply with it. (However, enforcement has historically been weak: it turned out that VISA had known about TJX's compliance problems, and had allowed them an extension until 2009 [1349].)

The latest implementations of WiFi are coming with mechanisms that encourage users to set up WPA encryption, and usability is a big deal for the other local connectivity protocols too.

21.4.5.3 Bluetooth

Bluetooth is another protocol used for short-range wireless communication. It's aimed at *personal area networks*, such as linking a headset to a mobile phone, or linking a mobile phone in your pocket to a hands-free phone interface in your car. It's also used to connect cameras and phones to laptops, keyboards to PCs and so on. Like WiFi, the initially deployed security protocol turned out to have flaws. In the original version, devices discover each other, and the users confirm that they wish two devices to pair by entering the same PIN at their keyboards. An attacker who's present during this pairing process can observe the traffic and then brute-force the PIN. Worse, Ollie Whitehouse, Yaniv Shaked and Avishai Wool figured out how to force two devices to rerun the pairing protocol, so that PIN-cracking attacks could be performed even on devices that were already paired [1341, 1156]. Denis Kügler also showed how to manipulate the frequency hopping so as to do a man-in-the-middle attack [747]. It's possible to mitigate these vulnerabilities by only doing pairing in a secure place and refusing requests to rekey.

Now, from version 2.1 (released in 2007), Bluetooth supports Secure Simple Pairing, an improved protocol [802]. This uses elliptic curve Diffie-Hellmann key exchange to thwart passive eavesdropping attacks, but man-in-the-middle attacks are harder; they are dealt with by generating a six digit number for numerical comparison, with a view to reducing the chance of an attack succeeding to one in a million. However, because one or both of the devices might lack a keyboard or screen (or both), it's also possible for the six-digit number to be generated at one device and entered as a passkey at another; and there's a 'just works' mode that's fully vulnerable to a middleperson attack. Finally, there's a capability to load keys out of band, such as from some other protocol that the devices use.

21.4.5.4 HomePlug

HomePlug is a protocol used for communication over the mains power line. An early version had a low bitrate, but HomePlug AV, available from 2007,

supports broadband. (Declaration of interest: I was one of the protocol's designers.) It aims to allow TVs, set-top boxes, personal video recorders, DSL and cable modems and other such devices to communicate in the home without additional cabling. We were faced with the same design constraints as the Bluetooth team: not all devices have keyboards or screens, and we needed to keep costs low. After much thought we decided to offer only two modes of operation: secure mode, in which the user manually enters into her network controller a unique AES key that's printed on the label of every device, and robust or 'simple connect' mode in which the keys are exchanged without authentication. In fact, the keys aren't even encrypted in this mode; its purpose is not to provide security but to prevent wrong associations, such as when your speakers wrongly get their audio signal from the apartment next door.

We considered offering a public-key exchange protocol, as with Bluetooth, but came to the conclusion that it didn't achieve much. If there's a middleperson attack going on where the attacker knocks out your set-top box using a jammer and connects a bogus box of the same type to your mains, then the chances are that you'll go to your network controller (some software on your PC) and see a message 'Set-top box type Philips 123 seeks admission to network. Certificate hash = 12345678. Admit/deny?' In such a circumstance, most people will press 'admit' and allow the attacker in. The only way to prevent them is to get them to read the certificate hash from the device label and type it in — and if they're going to do that, they might as well type in the key directly [967]. In short, our design was driven by usability, and we weren't convinced that public-key crypto actually bought us anything.

Time will tell which approach was best. And if we turn out to have been wrong, HomePlug (like Bluetooth and the latest versions of WiFi) lets keys be set up from other protocols by out-of-band mechanisms. So all devices in the office or home could end up with their keys managed by a single mechanism or device; and this could be convenient, or a source of vulnerabilities, depending on how future security engineers build it.

21.4.5.5 IPsec

Another approach is to do encryption and/or authentication at the IP layer using a protocol suite known as IPsec. IPsec defines a *security association* as the combination of keys, algorithms and parameters used to protect a particular packet stream. Protected packets are either encrypted or authenticated (or both); in the latter case, an authentication header is added that protects data integrity using HMAC-SHA1, while in the former the packet is encrypted and encapsulated in other packets. (The use of encryption without authentication is discouraged as it's insecure [151].) There's also an *Internet Key Exchange* (IKE) protocol to set up keys and negotiate parameters. IKE has been through a number of versions (some of the bugs that were fixed are discussed in [465]).

IPsec is widely used by firewall vendors who offer a *virtual private network* facility with their products; that is, by installing one of their boxes in each branch between the local LAN and the router, all the internal traffic can pass encrypted over the Internet. Individual PCs, such as workers' laptops and home PCs, can in theory join a VPN given a firewall that supports IPsec, but this is harder than it looks. Compatibility has been a major problem with different manufacturers' offerings just not working with each other; although firewall-to-firewall compatibility has improved recently, getting random PCs to work with a given VPN is still very much a hit-or-miss affair.

IPsec has the potential to stop some network attacks, and be a useful component in designing robust distributed systems. But it isn't a panacea. Indeed, virtual private networks exacerbate the 'deperimeterization' problem already discussed. If you have thousands of machines sitting in your employee's homes that are both in the network (as they connect via a VPN) and connected to the Internet (as their browser talks to the Internet directly via the home's cable modem) then they become a potential weak point. (Indeed, the U.S. Department of Justice ruled in 2007 that employees can't use their own PCs or PDAs for work purposes; all mobile devices used for departmental business must be centrally managed [108].)

21.4.5.6 TLS

Recall that when discussing public key encryption, I remarked that a server could publish a public key *KS* and any web browser could then send a message *M* containing a credit card number to it encrypted using *KS*: $\{M\}_{KS}$. This is in essence what the TLS protocol (formerly known as SSL) does, although in practice it is more complicated. It was developed to support encryption and authentication in both directions, so that both `http` requests and responses can be protected against both eavesdropping and manipulation. It's the protocol that's activated when you see the padlock on your browser toolbar.

Here is a simplified description of the version as used to protect web pages that solicit credit card numbers:

1. the client sends the server a *client hello* message that contains its name *C*, a transaction serial number *C*#, and a random nonce N_C;
2. the server replies with a *server hello* message that contains its name *S*, a transaction serial number *S*#, a random nonce N_S, and a certificate *CS* containing its public key *KS*. The client now checks the certificate *CS* back to a root certificate issued by a company such as Verisign and stored in the browser;
3. the client sends a *key exchange* message containing a *pre-master-secret* key, K_0, encrypted under the server public key *KS*. It also sends a *finished* message with a message authentication code (MAC) computed on all the

messages to date. The key for this MAC is the *master-secret*, K_1. This key is computed by hashing the pre-master-secret key with the nonces sent by the client and server: $K_1 = h(K_{CS}, N_C, N_S)$. From this point onward, all the traffic is encrypted; we'll write this as $\{...\}_{KCS}$ in the client-server direction and $\{...\}_{KSC}$ from the server to the client. These keys are generated in turn by hashing the nonces with K_1.

4. The server also sends a *finished* message with a MAC computed on all the messages to date. It then finally starts sending the data.

$C \rightarrow S: \ C, C\#, N_C$
$S \rightarrow C: \ S, S\#, N_S, CS$
$C \rightarrow S: \ \{K_0\}_{KS}$
$C \rightarrow S: \ \{finished, MAC(K_1, everythingtodate)\}_{KCS}$
$S \rightarrow C: \ \{finished, MAC(K_1, everythingtodate)\}_{KSC}, \{data\}_{KSC}$

The design goals included minimising the load on the browser, and then minimising the load on the server. Thus the public key encryption operation is done by the client, and the decryption by the server; the standard encryption method (*ciphersuite*) uses RSA for which encryption can be arranged to be very much faster than decryption. (This was a wrong design decision as browsers generally have a lot more compute cycles to spare than servers; it has created a brisk aftermarket for crypto accelerator boards for web servers.) Also, once a client and server have established a pre-master-secret, no more public key operations are needed as further master secrets can be obtained by hashing it with new nonces.

The full protocol is more complex than this, and has gone through a number of versions. It supports a number of different ciphersuites, so that export versions of browsers for example can be limited to 40 bit keys — a condition of export licensing that was imposed for many years by the U.S. government. Other ciphersuites support signed Diffie-Hellman key exchanges for transient keys, to provide forward and backward secrecy. TLS also has options for bidirectional authentication so that if the client also has a certificate, this can be checked by the server. In addition, the working keys *KCS* and *KSC* can contain separate subkeys for encryption and authentication. For example, the most commonly used ciphersuite uses the stream cipher RC4 for the former and HMAC for the latter, and these need separate keys.

Although early versions of SSL had a number of bugs [1308], version 3 and later (called TLS since version 3.1) appear to be sound (but they have to be implemented carefully [189]). They are being used for much more than electronic commerce — an example being medical privacy [280]. In our local teaching hospital, clinical personnel were issued with smartcards containing TLS certificates enabling them to log on to systems containing patient records. This meant, for example, that researchers could access clinical data from home during an emergency, or from their university offices if doing research. TLS

has also been available as an authentication option in Windows from Windows 2000 onwards; you can use it instead of Kerberos if you wish.

Another application is in mail, where more and more mail servers now use TLS opportunistically when exchanging emails with another mail server that's also prepared to use it. This stops passive eavesdropping, although it leaves open the possibility of middleperson attacks. To stop them too, you need some means of authenticating the public keys you use, and that brings us to the topic of public-key certificates.

21.4.5.7 PKI

During the dotcom boom, a number of companies achieved astronomical valuations by cornering the market in public-key certificates. The leading European certificate provider, Baltimore, achieved an eleven-figure market cap before crashing and burning in 2001. Investors believed that every device would need a public-key certificate in order to connect to other devices; you'd need to pay Baltimore (or Thawte, or Verisign) ten bucks every two years to renew the certificate on your toaster, or it wouldn't talk to your fridge.

As I discussed above, the keys in devices like fridges and toasters are best set up by local mechanisms such as the Bluetooth and HomePlug pairing mechanisms. But public key infrastructures are still used in a number of applications. First, there are the certificates that web sites use with TLS and that activate the security icon in your browser. Second, there are private infrastructures, such as those used by banks to set up keys for SWIFT, by mobile phone companies to exchange messages between Home Location Registers, and by companies that use TLS to authenticate users of their networks.

There is frequent semantic confusion between 'public (key infrastructure)' and '(public key) infrastructure'. In the first, the infrastructure can be used by whatever new applications come along; I'll call this an *open PKI*. In the second, it can't; I'll call this a *closed PKI*.

PKI has a number of intrinsic limitations, many of which have to do with the first interpretation — namely that the infrastructure is provided as a public service that anyone can use. I discussed many of the underlying problems in Chapter 7. Naming is difficult, and a certificate saying 'Ross Anderson has the right to administer the machine foo.com' means little in a world with dozens of people (and machines) of that name. Also, there's Kent's law: the more applications rely on a certificate, the shorter its useful life will be.

This is the 'one key or many' debate. As the world goes digital, should I expect to have a single digital key to replace each of the metal keys, credit cards, swipe access cards and other tokens that I currently carry around? Or should each of them be replaced by a different digital key? Multiple keys protect the customer: I don't want to have to use a key with which I can remortgage my house to make calls from a payphone. It's just too easy to

dupe people into signing a message by having the equipment display another, innocuous, one[2].

However, the killer turned out to be business needs. Multiple keys are more convenient for business, as sharing access tokens can lead to greater administrative costs and liability issues. There were many attempts to share keys; the smartcard industry tried to market 'multifunction smartcards' through the 1990s that could work as bank cards, electricity meter cards and even building access cards. Singapore even implemented such a scheme, in which even military ID doubled as bank cards. However, such schemes have pretty well died out. In one that I worked on — to reuse bank cards in electricity meters — the issues were control of the customer base and of the process of developing, upgrading and reissuing cards. In other cases, projects foundered because no-one could agree which company's logo would go on the smartcard.

Now the standard PKI machinery (the X.509 protocol suite) was largely developed to provide an electronic replacement for the telephone book, so it tends to assume that everyone will have a unique name and a unique key in an open PKI architecture. Governments hoped for a 'one key fits all' model of the world, so they could license and control the keys. But, in most applications, the natural solution is for each business to run its own closed PKI, which might be thought of at the system level as giving customers a unique account number which isn't shared with anyone else. Since then, the CA market has fractured; whereas in the late 1990s, Internet Explorer shipped with only a handful of CA keys (giving huge if temporary fortunes to the firms that controlled them), now the version in Windows XP contains hundreds.

This in turn leads to issues of trust. You don't really know who controls a key whose signature is accepted by a typical browser. Further issues include

- If you remove one of the 200-plus root certificates from Windows XP Service Pack 2, then Windows silently replaces it — unless you've got the skill to dissect out the software that does this [613]. Vista comes with fewer root certificates — but you can't delete them at all. This could be bad news for a company that doesn't want to trust a competitor, or a government that doesn't want to trust foreigners. For example, the large CA Verisign also does wiretap work for the U.S. government; so if I were running China, I wouldn't want any Chinese PC to trust their certificates (as Verisign could not just sign bad web pages — they could also sign code that Chinese machines would install and run).
- Usability is dreadful, as many sites use out-of-date certs, or certs that correspond to the wrong company. As a result, users are well trained to ignore security warnings. For example, when a New Zealand bank

[2]I just don't know how to be confident of a digital signature I make even on my own PC — and I've worked in security for over fifteen years. Checking all the software in the critical path between the display and the signature software is way beyond my patience.

messed up its certificate with the result that users got warned it didn't correspond to the bank, only one user out of 300 stopped — the rest just went ahead with their business [569].

- It's bad enough that the users don't care whether certificates work; yet the CAs don't seem to care, either. The UK certifier Tscheme was set up by industry as a self-regulatory scheme under the Electronic Communications Act as *'a source of independent assurance for all types of e-business and e-government transactions — especially for those transactions that depend on reliable, secure online identities.* It was noticed in July 2006 that `https://www.tscheme.org/` had its certification path misconfigured: there was a certificate missing in the middle of the chain, so verification failed unless you manually added the missing cert. By December 2007, it still wasn't properly fixed. According to the documentation, the 'HMG Root CA' should certify everything, yet it doesn't certify the Tscheme 'Trustis FPS Root CA', and neither is included in the standard Firefox distribution. In the CA world, it seems, everyone wants to be root, and no-one wants anyone else's signature on their keys, as then they'd have no reason to exist. So stuff still doesn't work.
- Many users disable security features on their browsers, even if these weren't disabled by default when the software shipped. Recall that the third step of the TLS protocol was for the client browser to check the cert against its stored root certificates. If the check fails, the browser may ask the client for permission to proceed; but many browsers are configured to just proceed anyway.
- Certs bind a company name to a DNS name, but their vendors are usually not authorities on either; they hand out certificates after cursory due diligence, and their 'certification practice statements' they go out of their way to deny all liability.
- There are still technical shortcomings. For example, the dominant certificate format (X.509) does not have the kind of flexible and scalable 'hot card' system which the credit card industry has evolved, but rather assumes that anyone relying on a cert can download a *certificate revocation list* from the issuing authority. Also, certs are designed to certify names, when for most purposes one wants to certify an authorization.

Behind all this mess lies, as usual, security economics. During the dotcom boom in the 1990s, the SSL protocol (as TLS then was) won out over a more complex and heavyweight protocol called SET, because it placed less of a burden on developers [72]. This is exactly the same reason that operating systems such as Windows and Symbian were initially developed with too little security — they were competing for pole position in a two-sided market.

The downside in this case was that the costs of compliance were dumped on the users — who are unable to cope [357].

In short, while public key infrastructures can be useful in some applications, they are not the universal solution to security problems that their advocates claimed in the late 1990s. It's a shame some governments still think they can use PKI as a mechanism for empire-building and social control.

21.5 Topology

The topology of a network is the pattern in which its nodes are connected. The Internet classically is thought of as a cloud to which all machines are attached, so in effect every machine is (potentially) in contact with every other one. So from the viewpoint of a flash worm that propagates from one machine to another directly, without human intervention, and by choosing the next machine to attack at random, the network can be thought of as a fully connected graph. However, in many networks each node communicates with only a limited number of others. This may result from physical connectivity, as with PCs on an isolated LAN, or with a camera and a laptop that are communicating via Bluetooth; or it may come from logical connectivity. For example, when a virus spreads by mailing itself to everyone in an infected machine's address book, the network that it's infecting is one whose nodes are users of the vulnerable email client and whose edges are their presence in each others' address books.

We can bring other ideas and tools to bear when the network is in effect a *social network* like this. In recent years, physicists and sociologists have collaborated in applying thermodynamic models to the analysis of complex networks of social interactions; the emerging discipline of network analysis is reviewed in [965], and has been applied to disciplines from criminology to the study of how new technologies diffuse.

Network topology turns out to be important in service denial attacks. Rulers have known for centuries that when crushing dissidents, it's best to focus on the ringleaders; and when music industry enforcers try to close down peer-to-peer file-sharing networks they similarly go after the most prominent nodes. There's now a solid scientific basis for this. It turns out that social networks can be modelled by a type of graph in which there's a power-law distribution of vertex order; in other words, a small number of nodes have a large number of edges ending at them. These well-connected nodes help make the network resilient against random failure, and easy to navigate. Yet Reka Albert, Hawoong Jeong and Albert-László Barabási showed that they also make such networks vulnerable to targeted attack. Remove the well-connected nodes, and the network is easily disconnected [20].

This has prompted further work on how topology interacts with conflict. For example, Shishir Nagaraja and I extended Albert, Jeong and Barabási's work to the dynamic case in which, at each round, the attacker gets to destroy some nodes according to an attack strategy, and the defender then gets to replace a similar number of nodes using a defense strategy. We played attack and defense strategies off against each other; against a decapitation attack, the best defense we found was a cell structure. This helps explain why peer-to-peer systems with ring architectures turned out to be rather fragile — and why revolutionaries have tended to organise themselves in cells [924]. George Danezis and Bettina Wittneben applied these network analysis ideas to privacy, and found that by doing traffic analysis against just a few well-connected organisers, a police force can identify a surprising number of members of a dissident organisation. The reason is that if you monitor everyone who calls, or is called by, the main organisers of a typical social network, you get most of the members — unless effort was expended in organising it in a cell structure in the first place [345].

These techniques may well become even more relevant to network attack and defence for a number of reasons. First, early social-network techniques have produced real results; the capture of Saddam Hussein used a layered social network analysis [623]. Second, as people try to attack (and defend) local networks organised on an ad-hoc basis using technologies like WiFi and Bluetooth, topology will matter more. Third, social networking sites — and more conventional services like Google mail that use introductions to acquire new customers — have a lot of social network information that can be used to track people; if a phisherman uses Google mail, the police can look for the people who introduced him, and then for everyone else they introduced, when searching for contacts. Fourth, as social structure starts to be used against wrongdoers (and against citizens by repressive regimes) people will invest in cell-structured organisations and in other stealth techniques to defeat it. Finally, there are all sorts of useful things that can potentially be done with topological information. For example, people may be more able to take a view on whether devices that are 'nearby' are trustworthy, and you may not need to filter traffic so assiduously if it can come from only a few sources rather than the whole Internet. This isn't entirely straightforward, as systems change over time in ways that undermine their builders' trust assumptions, but it can still be worth thinking about.

21.6 Summary

Preventing and detecting attacks that are launched over networks, and particularly over the Internet, is probably the most newsworthy aspect of security engineering. The problem is unlikely to be solved any time soon, as many

different kinds of vulnerability contribute to the attacker's toolkit. Ideally, people would run carefully written code on trustworthy platforms. In real life, this won't happen always, or even often. In the corporate world, there are grounds for hope that firewalls can keep out the worst of the attacks, careful configuration management can block most of the rest, and intrusion detection can catch most of the residue that make it through. Home users are less well placed, and most of the machines being recruited to the vast botnets we see in action today are home machines attached to DSL or cable modems.

Hacking techniques depend partly on the opportunistic exploitation of vulnerabilities introduced accidentally by the major vendors, and partly on techniques to social-engineer people into running untrustworthy code. Most of the bad things that result are just the same bad things that happened a generation ago, but moved online, on a larger scale, and with a speed, level of automation and global dispersion that leaves law enforcement wrong-footed.

Despite all this, the Internet is not a disaster. It's always possible for a security engineer, when contemplating the problems we've discussed in this chapter, to sink into doom and gloom. Yet the Internet has brought huge benefits to billions of people, and levels of online crime are well below the levels of real-world crime. I'll discuss this in more detail when we get to policy in Part III; for now, note that the $200 m–$1 bn lost in the USA to phishing in 2006 was way below ordinary fraud involving things like checks, not to mention the drug trade or even the trade in stolen cars.

Herd effects matter. A useful analogy for the millions of insecure computers is given by the herds of millions of gnu that once roamed the plains of Africa. The lions could make life hard for any one gnu, but most of them survived for years by taking shelter in numbers. The Internet's much the same. There are analogues of the White Hunter, who'll carefully stalk a prime trophy animal; so you need to take special care if anyone might see you in these terms. And if you think that the alarms in the press about 'Evil Hackers Bringing Down the Internet' are somehow equivalent to a hungry African peasant poaching the game, then do bear in mind the much greater destruction done by colonial ranching companies who had the capital to fence off the veld in 100,000-acre lots. Economics matters, and we are still feeling our way towards the kinds of regulation that will give us a reasonably stable equilibrium. In fact, colleagues and I wrote a report for the European Network and Information Security Agency on the sort of policy incentives that might help [62]. I'll return to policy in Part III.

Research Problems

Seven years ago, the centre of gravity in network security research was technical: we were busy looking for new attacks on protocols and applications

as the potential for denial-of-service attacks started to become clear. Now, in 2007, there are more threads of research. Getting protocols right still matters and it's unfortunate (though understandable in business terms) that many firms still ship products quickly and get them right later. This has led to calls for vendor liability, for example from a committee of the UK Parliament [625]. On the security-economics front, there is much interesting work to be done on decent metrics: on measuring the actual wickedness that goes on, and feeding this not just into the policy debate but also into law enforcement.

Systems people do a lot of work on measuring the Internet to understand how it's evolving as more and more devices, people and applications join in. And at the level of theory, more and more computer scientists are looking at ways in which network protocols could be aligned with stakeholder interests, so that participants have less incentive to cheat [971].

Further Reading

The early classic on Internet security was written by Steve Bellovin and Bill Cheswick [157]; other solid books are by Simson Garfinkel and Eugene Spafford on Unix and Internet security [517], and by Terry Escamilla on intrusion detection [439]. These give good introductions to network attacks (though like any print work in this fast-moving field, they are all slightly dated). The seminal work on viruses is by Fred Cohen [311], while Java security is discussed by Gary McGraw and Ed Felten [859] as well as by LiGong (who designed it) [539]. Eric Rescorla has a book on the details of TLS [1070]; another useful description — shorter than Eric's but longer than the one I gave above — is by Larry Paulson [1010]. Our policy paper for ENISA can be found at [62].

It's important to know a bit about the history of attacks — as they recur — and to keep up to date with what's going on. A survey of security incidents on the Internet in the late 1990s can be found in John Howard's thesis [626]. Advisories from CERT [321] and bugtraq [239] are one way of keeping up with events, and hacker sites such as `www.phrack.com` bear watching. However, as malware becomes commercial, I'd suggest you also keep up with the people who measure botnets, spam and phishing. As of 2007, I'd recommend Team Cymru at `http://www.cymru.com/`, the Anti-Phishing Working Group at `http://www.antiphishing.org/`, the Shadowserver Foundation at `http://www.shadowserver.org/`, Symantec's half-yearly threat report at `www.symantec.com/threatreport/`, and our blog at `www.lightbluetouchpaper.net`.