**Chapter 21**

**Network Attack and**  
 **Defence**

**Simplicity is the ultimate sophistication.**

– Leonardo Da Vinci

**There’s no security here – keep moving!**

– Richard Clayton

**21.1** **Introduction**

In this chapter I’m going to try to draw together the network aspects of security  
 in a coherent framework. This is not straightforward as much of network security  
 is practical engineering; a purist from computer science might see the ﬁeld as one  
 bodge piled on top of another. And network security may not be that important  
 to many developers: if you write apps for Androids and iPhones that talk to  
 services on AWS or Azure, then you can leave much of the worry to Amazon or  
 Microsoft.

But many organisations need to pay attention to network security, and there

are some visible strategic trends. For twenty years, it was accepted that ﬁrms  
 would have a trusted internal network or intranet, protected from the Internet  
 by ﬁrewalls; while taken to extremes by defence and intelligence organisations  
 with classiﬁed internal networks, milder versions were seen as best practice  
 by most normal ﬁrms. And some industries have no viable alternatives. For  
 example, the protocols used in industrial control systems – DNP3 and Modbus  
 – don’t support encryption or authentication, as they evolved in the days of  
 leased lines and private radio links. By the late 1990s, control systems engineers  
 were attaching sensors and actuators to IP networks, as they were cheaper –  
 and then realising that anyone in the world who knew a sensor’s IP address  
 could read it, and anyone who knew an actuator’s IP address could activate  
 it. This led to the growth of specialist ﬁrms who sell ﬁrewalls that understand  
 these protocols; energy companies have thousands of them. A typical electricity

substation might have two hundred devices from a multiplicity of vendors, on  
 a LAN where performance is critical, so retroﬁtting crypto is impractical; but  
 it has one connection to the outside world, so that’s where you have to put the  
 protection. This is known as *re-perimeterization*. The same approach is taken  
 with vehicles, where the internal CANBUS cannot be protected, so the radio  
 interfaces with the outside world have to be.

But in many ﬁrms the trend is ﬁrmly in the other direction, towards *de-*

*perimeterisation*. One thought leader is Google, promoting an architecture

without ﬁrewalls which it calls a *zero-trust security model*: “By shifting access  
 controls from the network perimeter to individual users and devices, Beyond-  
 Corp allows employees, contractors, and other users to work more securely from  
 virtually any location without the need for a traditional VPN.” Google’s ex-  
 perience is that the move to mobile and cloud technology is making network  
 perimeters ever harder to deﬁne, let alone police, and if a ﬁrm’s large enough  
 that some internal compromise is inevitable anyway then the perimeter is the  
 wrong place to put the primary protection [1984]. There are still some perimeter  
 defences, most notably against service-denial attacks, but internal networks are  
 otherwise unprivileged and the emphasis is on tight authentication and autho-  
 risation of users and devices: each service has an Internet-facing access proxy.  
 One might see this as a per-service ﬁrewall rather than a per-building ﬁrewall,  
 but there is quite a lot more to it with tiers of sensitivity, a device inventory  
 service and an access control engine [1479]. You also need really good HR data,  
 so you can tie staff and contractors to devices and the services they’re allowed  
 to use. Much the same architecture is being adopted by other ﬁrms operating  
 large-scale data centres, and zero-trust security is now the subject of draft stan-  
 dards activity by NIST [1618]. It will no doubt get a boost from the pandemic  
 because of the huge increase in home working.

Other organisations may take a hybrid approach. The university where I

work, for example, has some defences at the perimeter but largely lets depart-  
 ments do our own thing; a computer science department has quite different  
 requirements from a humanities department or the ﬁnance office.

In order to explore the options and constraints, I’m ﬁrst going to discuss net-

working protocols such as BGP, DNS and SMTP and the service-denial attacks  
 that can result from their abuse. I’ll then take a closer look at malware, and  
 then at defensive technologies such as ﬁltering and intrusion detection and how  
 defenders can coordinate them. I’ll then survey the limitations of widely-used  
 crypto protocols such as TLS, SSH and IPsec, and the particularly tricky role  
 of certiﬁcation authorites. Finally I’ll return to network architecture. Many is-  
 sues are complex and interlinked, with some signiﬁcant trade-offs. For example,  
 various kinds of end-to-end crypto can bring beneﬁts – particularly against bulk  
 surveillance – but can get in the way of the surveillance we want for network  
 security.

This chapter will deal with ﬁxed networks, and I’ll discuss what’s different

about mobile networks in the following chapter.

|  |  |  |
| --- | --- | --- |
|  |  |  |

**21.2** **Network Protocols and Service Denial**

I’m going to assume some familiarity with basic network protocols. The tele-  
 graphic 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. ISPs are migrating to IP version 6, as the 4 billion possible IPv4  
 addresses are just about allocated; IPv6 uses 128-bit addresses. Some 10–15%  
 of traffic is now IPv6; in many countries a new broadband subscription will get  
 you an IPv6 address which works for all normal consumer purposes.

Local networks mostly use ethernet, in which devices have unique ethernet

addresses (also called MAC addresses) that are mapped to IPv4 addresses using  
 the *address resolution protocol* (ARP). The *Dynamic Host Conﬁguration Pro-*  
 *tocol* (DHCP) is used to allocate IP addresses to machines as needed and to  
 ensure that each IP address is unique. *Network address translation* (NAT) also  
 enables multiple devices on a network to use the same Internet-facing IP ad-  
 dress, typically with different port numbers; this is used by most mobile network  
 operators and many ISPs. 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  
 of devices to IP addresses. There may be more than one log, and lots can go  
 wrong – such as wrong timestamps, and failure to understand time zones.

One of the most basic concerns is the prevention and mitigation of *denial-of-*

*service* (DoS) attacks. These have a number of ﬂavours. An opponent can try  
 to steal some of your IP address space, or one or more of your domains, in order  
 to send spam; even when you get it back, you may ﬁnd it’s been extensively  
 blacklisted. An opponent can send you huge ﬂoods of traffic from a botnet

of many compromised machines; a *distributed denial-of-service* (DDoS) attack.  
 They can abuse various online services such as DNS to send you ﬂoods of packet  
 traffic. Let’s work through these in turn.

**21.2.1** **BGP security**

The Internet is an interconnected network of networks: its components are  
 *Autonomous Systems* (ASes) such as ISPs, telcos and large organisations, each  
 of which controls a range of IP addresses. The glue that holds them together,  
 the core routing protocol of the Internet, is the *Border Gateway Protocol* (BGP).  
 Routers – the specialized computers that switch packets on networks – 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 to use. ASes can route traffic to other ASes by buying service from large  
 transit providers but typically cut the costs of this by peering with each other  
 at a local *Internet interchange* (IX), of which most countries have at least one  
 and large countries may have several.

Internet interconnectivity is a complex ecosystem with many interdependent

layers. Its open and decentralised organisation has been essential to the success  
 and resilience of the Internet, which has meant that the effects of natural dis-  
 asters such as Hurricane Katrina and terrorist attacks such as 9/11 have been

|  |  |  |
| --- | --- | --- |
|  |  |  |

limited in time and space, as have assorted technical failures. However the In-  
 ternet is slowly becoming more centralised, as a result of the consolidation of  
 Tier-1 providers, and is vulnerable to common-mode failures (such as electric  
 power cuts) as well as to disruptive attacks.

About the worst attack we can reasonably foresee would involve an attacker

planting malware on thousands of routers so they advertise large numbers of false  
 routes, clogging the routing tables and tearing up the routing fabric. There have  
 been several warnings already in the form of incidents and accidents. In 2008,  
 YouTube became inaccessible for a few hours after the government of Pakistan  
 tried to censor it locally by announcing false routes to it, which propagated  
 globally; and in 2010 China Telecom advertised over 100,000 invalid routes,  
 hijacking 15% of Internet addresses for 18 minutes. Some people ascribed that  
 to accident, while others suggested that China had been testing a ‘cyber-nuke’,  
 some of whose fallout escaped. Most routers now accept only a limited number  
 of routes from each of their peers, be it a few dozen or a few hundred; so large-  
 scale disruption would require thousands of subverted routers. Both China

and (more recently) Russia have been working on making the Internet in their  
 countries separable, so that major disruptive attacks could in theory be launched  
 without inﬂicting unacceptable collateral damage on local services and facilities.  
 There have been reports of BGP hijacking being used by China for intelligence  
 collection; for example, traffic from Canada to Korean government websites was  
 routed via China from February 2016 for six months [533]. There has also been  
 criminal misuse, ranging from the hijacking of IP address space by spammers, to  
 an eight-ﬁgure ad fraud in 2018 whose perpetrators hid in address space stolen  
 from the US Air Force [791]. Finally, there is a growing political tussle in 2019–  
 20 about whether Huawei should be allowed to sell routers at scale (or at all)  
 in countries allied to the USA.

Taking a step backward, the resilience of the Internet is hard to deﬁne and to

measure; it is in tension with efficiency and may be decreasing as a small number  
 of very large networks come to dominate. These range from the dominant transit  
 provider, Level 3, to content delivery networks (CDNs) operated by Google,  
 Akamai, Cloudﬂare and others. There are many complex interactions between  
 resilience and efficiency, reachability and congestion, traffic prioritisation and  
 commercial sensitivity, complexity and scale. There’s no mechanism to check  
 the validity of routing information distributed via BGP. The pervasive mistrust  
 between ISPs and governments makes regulation difficult. The lack of good

information about how the system works makes rational discussion difficult too.  
 Resilience has so far depended on surplus capacity and rapid growth, but that  
 cannot continue for ever. In 2011 colleagues and I wrote a major report for the  
 European Network and Information Security Agency that explores these issues  
 in detail [1906].

The main technical BGP security mechanism at present is the *Resource*

*Public Key Infrastructure* (RPKI) which enables registries to certify that “Au-  
 tonomous system X announces IP address range Y”. This will not prevent ca-  
 pable attackers, as a malicious route announcement will just have the right AS  
 at the end of the route following the attacker’s in the middle; but it detects  
 the fat-ﬁnger mistakes that cause most of the outages. Whether it will make  
 an already fragile BGP system more robust to have lots of certiﬁcates in it re-

|  |  |  |
| --- | --- | --- |
|  |  |  |

mains to be seen; when RIPE’s certiﬁcate expired in February 2020 there was a  
 short outage until it was ﬁxed. For the future, people are working on Peerlock,  
 whereby the main ASes at an interchange share information about what routes  
 they will and won’t announce; this has the prospect of bringing enough local  
 beneﬁt to exchange members for it to be practically deployable.

**21.2.2** **DNS security**

The *Domain Name System* (DNS) allows mnemonic names such as ross-anderson.  
 com to be mapped to IP addresses of either kind; there’s a hierarchy of DNS  
 servers that do this, ranging from several hundred top-level servers down through  
 machines at ISPs and on local networks, which cache DNS records for perfor-  
 mance and reliability. It does occasionally get attacked: the Mirai botnet at-  
 tacked DynDNS in October 2016, taking out Twitter on the US eastern seaboard  
 for ﬁve hours. But DNS has become a massively distributed system with lot of  
 very fast machines connected to very high-capacity networks, so service denial  
 attacks on it are rare.

Hijacking does occur from time to time, and at various levels. Some states

intercept and redirect DNS queries as a means of censorship; some ISPs have  
 done so, as a means of replacing ads in web pages with ads from which they  
 get a cut; and a DNS server at an ISP may be hacked to drive clients to a  
 wicked website. This is known as *pharming*, and in a variant called *drive-by*  
 *pharming*, the crooks lure you to a web page containing javascript that changes  
 your home router’s DNS server from the one at your ISP to one under their  
 control [1816]. Next time you try to go to www.citibank.com, you may be

directed to a phishing site that emulates it. That’s one reason to change the  
 default password on your home router – even if it’s only accessible from inside  
 your network.

In order to prevent DNS hijacking, DNSSEC adds digital signatures to DNS

name records. By verifying such a signature you can check that the record

came from the authoritative server and was not altered en route. Uptake is  
 patchy: all US government domains in .gov are supposed to be signed, and most  
 domains in Sweden are signed, as the registrar made signed domains cheaper.  
 However some major ﬁrms like Google don’t sign their DNS records out of  
 concern that cryptography makes systems more fragile; if anything goes wrong,  
 you can just disappear. Other ﬁrms avoid DNSSEC because they don’t want  
 competitors to ‘walk the zone’ and enumerate all their subdomains; the NSEC3  
 extension enables ﬁrms to avoid this using hashes, but many ﬁrms (or their  
 service providers) have not yet built the infrastructure.

Another problem with DNSSEC is that it gets abused in denial-of-service

attacks. A common technique is that Alice attacks Bob by sending Charlie

a message saying, “Hey, can you tell me the very large answer to this short  
 question? Yours, Bob!” As signed DNS records are a lot larger, a DDoS-for-hire  
 service can use DNSSEC as an ampliﬁer, Alice can send packets that purport  
 to come from Bob’s IP address to many DNS servers, which then bombard the  
 target with replies. (Cheeky criminals use the FBI as Charlie, as fbi.gov has  
 two nice big keys.)

|  |  |  |
| --- | --- | --- |
|  |  |  |

The controversial issue in 2020 is *DNS-over-https* (DoH). The main browser

maintainers, Chrome and Mozilla, propose that rather than sending DNS traffic  
 in the clear, it will go encrypted over https to a DoH resolver. This is claimed  
 to be good for privacy, as your ISP will have less information about your brows-  
 ing (but unless you use Tor, it will still have plenty). The downside is that  
 many enterprise security products monitor DNS to detect abuse. If malware  
 compromises a machine in your ﬂeet, you may spot it when it tries to contact  
 a command-and-control server, so enterprises buy threat intelligence feeds and  
 monitor the domain names (and IP addresses) blacklisted on them. Sysadmins  
 also like to monitor for DNS hijacking, and to block certain domains as inappro-  
 priate for work. DoH will make all this harder, and is questionable architecture  
 as running a core network service over an application means it’s ‘not the Inter-  
 net any more’ [428]. On the commercial side, DoH may entrench Google’s grip  
 on the advertising market, while causing problems for content delivery networks  
 like Akamai and Cloudﬂare over routing, load balancing and so on. It will also  
 stop ISPs transcoding videos for mobile users to save bandwidth. Experts would  
 have preferred to run DNS over TLS instead.

**21.2.3** **UDP, TCP, SYN ﬂoods and SYN reﬂection**

On wide-area networks, most data move between machines using either the *User*  
 *Datagram Protocol* (UDP) which is connectionless, or the *Transmission Control*  
 *Protocol* (TCP) which sets up persistent connections between endpoints. Let’s  
 start with the 3-way handshake used by Alice to initiate a TCP connection to  
 Bob and set up sequence numbers for subsequent packet traffic.

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

This protocol has been exploited in a many ways. The classic service-denial

attack is the *SYN ﬂood*. Alice simply sends a lot of SYN packets and never  
 acknowledges any of the replies. Bob accumulates more records of SYN packets  
 than his software can handle. This was used in one of the ﬁrst distributed

denial-of-service attacks that brought down Panix, a New York ISP, for several  
 days in 1996.

The technical ﬁx was the ‘SYNcookie’: rather than keeping a copy of the

incoming SYN packet, *B* simply sends out as *Y* an encrypted version of *X*. That  
 way, Bob doesn’t have to retain a lot of state about half-open sessions. Despite  
 this, SYN ﬂoods persisted, albeit at a declining rate, for many years. The

general principle is that when you’re designing a protocol anyone can invoke,  
 don’t let malicious users force honest ones to do work.

The more common attack now is *SYN reﬂection*. Alice sends Bob a packet

|  |  |  |
| --- | --- | --- |
|  |  |  |

that purports to come from Charlie. Bob replies to Charlie, and in practice  
 systems send up to ﬁve ACKs in response to each SYN as a robustness measure,  
 so there’s still a useful ampliﬁcation effect.

**21.2.4** **Other ampliﬁers**

Many other protocols have been used in service-denial attacks than DNS and  
 TCP [1503]. An early favourite 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. If Alice sent an ICMP packet purporting to come  
 from Bob to a broadcast address, all the the machines on the subnet would send  
 him a response. The protocol was changed so that broadcast addresses didn’t  
 reply. The bad guys changed to use protocols such as NTP and DNS for which  
 ampliﬁers could still be found.

More thorough ﬁxes for attacks based on packet ampliﬁcation were to follow.

Most of the available ampliﬁers use UDP packets, including ICMP and NNTP  
 but not SYN reﬂection; so starting from the mid-2000s, broadband ISPs started  
 ﬁltering out UDP packets with forged source addresses. Microsoft also 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. So attacks that exploit UDP packet ampliﬁers have  
 to be run from servers in hosting centres. In the late 2010s, such attacks have  
 become increasingly the preserve of DDoS-for-hire operators, against whom the  
 most effective countermeasure has been to raid them and arrest them.

**21.2.5** **Other 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, by sending ﬂoods of  
 packets from infected machines. The ﬁrst *distributed denial of service* (DDoS)  
 attack may have been the Morris worm in the 1980s, and the ﬁrst deliberate one  
 in the 1990s with the attack already mentioned on Panix. Nowadays, botnets  
 are assembled using all sorts of vulnerabilities, and underground markets let  
 some people specialise in hacking machines and selling them to others who  
 extract value in various ways. Since 2016, the machines most used for DDoS  
 have been IoT devices such as CCTV cameras, which are now connected in large  
 numbers to home WiFi networks with reasonable bandwidth, but which tend to  
 have known default passwords – and are often incapable of being patched. The  
 Mirai botnet appeared in October 2016 to exploit this opportunity, and there  
 have been over a thousand variants of it since (its source code got posted to  
 Hackforums).

There are various motives for service-denial attacks. Most are launched

by schoolkids – typically gamers who want to take down an opposing crew’s  
 teamspeak server. There has for some years been a black market in DDoS-for-  
 hire, which the authorities in the USA and elsewhere have been trying to close  
 down. There have been some incidents of blackmail (e.g. of online bookmakers),  
 and a growing use of the technique for suppressing political opponents – starting

|  |  |  |
| --- | --- | --- |
|  |  |  |

perhaps with attacks on the servers of an opposition party in Kyrgyzstan, even  
 when these were relocated to North America [1613]. We discussed their use in  
 conﬂict by states in Chapter 2.

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 protest can easily  
 shade over into abuse, and drawing legislative distinctions can be hard.

**21.2.6** **Email – from spooks to spammers**

The SMTP standard for email has particular issues around the prevention of  
 bulk interception, and the prevention of bulk unwanted mail.

Email is by default neither encrypted nor authenticated, and was for decades

available to anyone who could either monitor the network or access mail servers.  
 It was possible to use programs such as PGP/GPG to encrypt mail, but this  
 never caught on outside small communities. First, such programs can be a

pain to use, and second, there are strong network effects: there’s no point in  
 using email encryption if none of your friends do. What’s more, if only a small  
 group of people use encryption, this may just bring them to the attention of  
 the authorities; subversive groups, spies and so on really need anonymity rather  
 than just conﬁdentiality, as we discussed in section 20.4. So PGP/GPG tends  
 to be used by specialists, such as sysadmins and anti-virus researchers.

There are two main countermeasures to bulk interception. First, most mail

servers use starttls to set up encrypted communications with other mail  
 servers as they exchange mail, especially since the Snowden revelations. En-  
 crypted exchanges can be blocked by man-in-the-middle attacks, and these have  
 been reported in some less-democratic countries. The current countermeasure  
 to such attacks, *MTA Strict Transport Security* (MTA-STS), is supported by  
 Microsoft, Google and Yahoo [1220]: it allows mail service providers to specify  
 that mail should only be delivered to them via a TLS session authenticated  
 by a proper certiﬁcate which you download from their website. This prevents  
 downgrade or interception attacks on email to and from the big boys, and also al-  
 lows opportunistic, trust-on-ﬁrst-use encryption to other servers. MTA-STS has  
 generally supplanted an earlier standard, *DNS-based Authentication of Named*  
 *Entities* (DANE) which put a TLS certiﬁcate for starttls in the mail server’s  
 DNS record1.

The second countermeasure is that some 95% of personal email accounts

nowadays are at the big ﬁve webmail providers, and many corporates use them  
 too. In this case, the conﬁdentiality of email is assured by TLS, fortiﬁed with  
 certiﬁcate pinning and certiﬁcate transparency which we’ll discuss later. But  
 although bulk access may be blocked, webmail is subject to warranted access,  
 just like other services that corporates outsource.

Bulk unwanted mail, or spam, has two components. The ﬁrst is entirely legal

but unwanted marketing communication. As marketers can make it tiresome to

1DANE is still widely used in Germany, but Google refused to use it as it depends on

DNSSEC which Google considers to be insufficiently dependable.

|  |  |  |
| --- | --- | --- |
|  |  |  |

opt out, users ﬁnd it more convenient to press the ‘report spam’ button once an  
 offer or supplier is no longer of interest.

The second consists of ﬂoods of generally unwanted traffic sent out for the

most part by botnets, and often with clear criminal intent. This is in some  
 respects similar to a DDoS attack: just as DDoS bots may forge IP addresses,  
 spam bots may forge the sender’s email address. This is fought by the big

providers with four main mechanisms.

1. *Domain Keys Identiﬁed Mail* (DKIM) ties email to the sending domain by

signing it using a signature key whose public veriﬁcation key is kept in the  
 sending domain’s DNS record. The signed material is selected to identify  
 the message unambiguously despite the additions to headers that occur  
 during transit, but to stop the bad guys adding an extra “From: PayPal”  
 header. Mail that hasn’t been altered too much can be forwarded. There’s  
 a replay attack in that the spammer sends his spam through Gmail, which  
 signs it, and then forwards it afterwards; so mail servers cache DKIM  
 signatures and discard mail carrying a signature that’s already been seen  
 a few times.

2. *Sender Policy Framework* (SPF) is similar but ties mail to the source IP

address. Again, this is veriﬁable against a key in the domain DNS record.  
 SPF doesn’t allow mail forwarding; mailing list servers are supposed to use  
 a related protocol called *Authenticated Received Chain* (ARC) to re-sign  
 mail they forward.

3. A domain’s DNS can also contain a *Domain-based Message Authentication,*

*Reporting and Conformance* (DMARC) record which enables its owner to  
 recommend what a recipient should do with email that appears to come  
 from the owner’s domain but which fails authentication using both DKIM  
 and SPF.

4. Machine-learning systems are used to ﬁlter mail against authentication re-

sults and other criteria, and take much of their ground truth from whether  
 users report mail as spam. This is made more complicated by user pref-  
 erences for marketing material, which vary by user and over time.

The illegal segment of spam is now a highly specialised business, run by

several large gangs. Its statistics have been ‘lumpy’ since the mid-2000s and  
 this has been getting more pronounced. As of 2020, the gangs typically steal IP  
 address space using malicious BGP route announcements, register thousands of  
 domains, and send a few hundred spams from each before the machine-learning  
 ﬁlters kick in and block them.

**21.3** **The Malware Menagerie – Trojans, Worms**  
 **and RATs**

The ﬁrst examples of malicious code were *Trojan Horses* – named after the  
 horse the Greeks left for the Trojans, supposedly as a gift but which contained

|  |  |  |
| --- | --- | --- |
|  |  |  |

soldiers who opened the gates of Troy to the Greek army [1129]. There have  
 been religious wars over nomenclature for years, which is why many people  
 prefer to just use the term *malware*. My usage is that a Trojan is a program  
 that does something malicious (such as capturing passwords) when run by an  
 unsuspecting user. A *worm* is a malicious program that replicates itself on other  
 systems, while one that does so by hooking itself into the code of other programs  
 is a *virus*. A *remote access Trojan* (RAT) is software that may or may not run  
 as root but that enables a remote party to access the device it runs on, while  
 a *rootkit* is software installed as root on a device and that stealthily enables a  
 third party to control it. *Potentially unwanted software* (PUS) may have been  
 installed openly or by deception, but does something the user doesn’t want (if  
 they understand it at all).

These categories are not mutually exclusive and the boundaries can be con-

text dependent. For example, stalkerware – software that enables one person  
 to track another’s mobile phone location and use – falls into different categories  
 depending on whether it was installed covertly, or by a controlling man bullying  
 his partner, or by a court ordering it as a condition of bail. Even stealthy mal-  
 ware isn’t always illegal as it can be used by law-enforcement agencies to turn  
 suspects’ phones and laptops into listening devices, as well as by fraudsters to  
 operate bank accounts by remote control2.

Malware generally uses stealth techniques to hide, but eventually it’s iden-

tiﬁed and tools to remove it are written. There’s a whole ecosystem around  
 malware: malware writers, botnets of infected machines, and a range of se-  
 curity ﬁrms offering everything from threat intelligence to antivirus software.  
 (There are even ﬁrms selling malware – particularly to government agencies.)  
 And in addition to the formal economy, there’s an underground economy of  
 cyber-crooks selling everything from banking Trojans to DDoS-for-hire services.

**21.3.1** **Early history of malware**

It the early 1960’s, machines were slow and their CPU cycles were rationed –  
 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 a privileged account with a known password. By the  
 1970s, 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 [1724].

In 1984, Ken Thompson gave a classic paper“Reﬂections On Trusting Trust”,

when he accepted a Turing award, the top prize in computer science. 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 [1883]. His trick  
 was to build the trapdoor into the compiler. If this recognized that it was

2At the other end of the spectrum, some antivirus products behave like malware in various

ways, including being very hard to remove after a ‘free trial’, or by introducing insecurities.  
 In December 2019, one brand of AV software was removed by Chrome, Firefox and Opera for  
 exﬁltrating too much personal information [358].

|  |  |  |
| --- | --- | --- |
|  |  |  |

compiling the login program, it would insert a master password that would  
 work on any account3. Of course, someone might examine the source code

for the compiler, and then compile it again from scratch. So if the compiler  
 recognizes that it’s compiling itself, it inserts the vulnerability anyway, even if  
 it’s not present in the source. So even if you can buy a system with veriﬁably  
 secure hardware, operating system and applications, the compiler binary can  
 still contain a Trojan. The moral is that in order to trust a system completely,  
 it is not enough to build all of it, in the sense that software engineers use the  
 word ‘build’, namely compiling it from source code. You have to create all of it,  
 including the tool chain, and the hardware too.

Malware next became mobile. The ﬁrst-ever computer virus in the wild was

written for the Apple II by a 9th-grader in 1981 [1216]. In 1984 Fred Cohen did a  
 PhD on the topic; his experiments with different operating systems showed how  
 code could propagate itself from one machine to another, and as I mentioned in  
 Section 9.6.4, from one compartment of a multilevel system to another. Within  
 about three years we started to see the ﬁrst real live viruses in the wild: PC  
 viruses which spread when users shared programs on diskettes or via bulletin  
 boards4.

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, and  
 inviting users to run it, it was ahead of its time.

**21.3.2** **The Internet worm**

The press and public became aware of malware in November 1988 with the In-  
 ternet worm. This was a program written by Robert Morris Jr that exploited a  
 number of vulnerabilities to spread from one machine to another in November  
 1988 [617]. It tried 432 common passwords in a guessing attack, looked for any  
 machines trusted by the machine it infected, and also tried to exploit vulnera-  
 bilities in Unix (including the fingerd bug mentioned in section 6.4.1). It also  
 took steps to camouﬂage itself: it was called sh and it encrypted its data strings  
 (albeit with a Caesar cipher).

Its author claimed that his code was not a deliberate attack on the Internet

– merely an experiment to see whether code could replicate from one machine to  
 another. But it had a bug. It should have recognised machines that were already  
 infected, and not infected them again, but this feature didn’t work. The result  
 was a huge volume of traffic that completely clogged up the Internet (or more  
 accurately, its predecessor the Arpanet) despite the fact that it only affected  
 some 10% of the 60,000 machines on the Arpanet at the time. One lesson

3This developed an idea ﬁrst ﬂoated by Paul Karger and Robert Schell in the Multics

evaluation in 1974 [1019].

4Before the Internet was opened up to the public, online services were mostly standalone;

bulletin boards were typically operated by hobbyists and would let subscribers or even anony-  
 mous users dial in to share information and ﬁles.

|  |  |  |
| --- | --- | --- |
|  |  |  |

was that sites which 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** **Further malware evolution**

By the early 1990s, PC viruses had become such a problem that they gave  
 rise to a whole industry of anti-virus software. Through the 1990s, operating  
 systems acquired better access controls, making the malware writer’s job harder,  
 but the spread of interpreted languages provided plenty of new opportunities.  
 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 [298].

The next phase of malware evolution was to enlist the user as the propagation

mechanism. The ‘Love Bug’ in 2000 was a worm that sent itself to everyone  
 in the victim’s address book, with the subject line ‘I love you’ designed to get  
 people to open it5. This incident taught us about the difficulty of stopping

such things by ﬁltering; a Canadian company with 85,000 staff stripped out  
 all Windows executables at the ﬁrewall, but many of their staff had personal  
 webmail accounts, so the Love Bug got in anyway. The company had given  
 each employee a copy of the corporate directory in their address book, and the  
 result was meltdown as 85,000 mail clients each tried to say ‘I love you’ to  
 each of 85,000 addresses. The Love Bug was followed by similar worms which  
 persuaded people to click on them by offering pictures of celebs such as Britney  
 Spears and Paris Hilton.

The next development was *ﬂash worms* which propagate by scanning the

whole Internet for machines vulnerable to some exploit or other, and taking  
 them over; examples such as Code Red and Slammer infected all vulnerable  
 machines within hours or even minutes, and drove research into what sort of  
 automated defences might react in time [1821].

The early 2000s also saw the rise of *spyware* and *adware*. Spyware collects

and forwards information from your computer (and now, your phone) without  
 the owner’s authorization, or with at best an obscure popup that doesn’t really  
 tell you what you’re agreeing to. It may also be installed by someone else,

such as a parent or partner; spyware is increasingly involved in intimate partner  
 abuse. Adware may bombard the user with advertising popups and can be

bundled with spyware. The vendors of such products have even sued antivirus  
 companies who blacklisted their wares. Some spyware is installed deliberately,  
 whether by companies who want to keep tabs on staff, by parents who want  
 to see what their kids are up to, or by abusive men who want to monitor and  
 control their partners. Boundaries are difficult and different people may have  
 different views.

A sea-change came about in 2004–6. Until then, most malware writers did

so for fun or to impress their friends – basically, they were amateurs. Since  
 then, the emergence of underground markets and crime forums has made the

5It can be seen as a more virulent variant of the ‘Christma’ worm of 1987, but the Love

Bug’s author was a schoolboy in Manila who’d probably never heard of that one.

|  |  |  |
| --- | --- | --- |
|  |  |  |

whole business much more professional. Malware writers now get paid money  
 for software to recruit machines that can be sold on for cash to botnet herders  
 and for other exploits.

Back in the amateur era, most viruses were ﬂaky; very few actually spread in

the wild. If code isn’t infectious enough it won’t spread, but if you make it too  
 infectious then within a few hours the world’s anti-virus vendors are upgrading  
 their products to detect and remove it. Now that malware writers focus on  
 money rather than bragging rights, they tend to avoid self-replicating worms  
 in favour of more controllable exploit campaigns. (The main exception is when  
 exploiting IoT devices that can’t be patched.)

By the late 2000s, the largest botnets were using professional online market-

ing techniques to grow their network. Various stories were used to get people to  
 click on a link and run a Trojan that would drop a rootkit on to their machine.  
 Victims had to click away several warnings to install software; but Windows  
 pops up so many annoying dialog boxes that most people just click them away.  
 One of the ﬁrst really large ones, Storm, earned its living from pump-and-dump  
 operators and pharmacy scammers [1090]. Security researchers tried to dis-

able big botnets by ﬁnding and taking down their command-and-control server;  
 Storm used a peer-to-peer architecture that removed this single point of fail-  
 ure [1835]. In the end, it was targeted by Microsoft for removal. The same  
 game is still being played; in March 2020 Microsoft took down Necurs, a botnet  
 with nine million machines that had been growing for eight years, distributing  
 banking Trojans as well as ransomware and email spam [349].

Flash worms have made a comeback since October 2016 with the Mirai worm

and its variants. Mirai initially took over wiﬁ-attached CCTV cameras that had  
 a known root password and software that could not be upgraded; all such devices  
 in the IPv4 address space could be found and recruited within an hour or so.  
 Since then, there have been over a thousand Mirai variants attacking various  
 IoT devices.

**21.3.4** **How malware works**

Malware typically has two components – a replication mechanism or dropper,  
 and a payload. A worm simply makes a copy of itself somewhere else when it’s  
 run, perhaps by breaking into another system by password guessing or using a  
 remote code execution vulnerability (both of which were used by the Internet  
 worm). Viruses spread in other software, perhaps as macros in documents, while  
 Trojans are typically executed by the victim.

The second component of a virus is the payload. When activated, this may

do one or more of a number of bad things:

*•* exﬁltrate your conﬁdential data;

*•* attack you directly using banking malware or spyware;

*•* encrypt your data and demand a ransom;

*•* attack others, such as when GCHQ’s Operation Socialist described in sec-

|  |  |  |
| --- | --- | --- |
|  |  |  |

tion 2.2.1.9 subverted Belgacom and installed software in it to do surveil-  
 lance of mobile-phone traffic passing through Belgium to other countries;

*•* perform some other nefarious task, such as using the CPU to mine cryp-

*•* install a rootkit or remote access Trojan to enable its controllers to do any  
 and to update itself in response to any countermeasures.

If the target is not an individual but a company – as in the Belgacom case

– then the attack may involve weeks to months of work. Once attackers control  
 a device on the target network, they will want to move sideways to map the  
 network and ﬁnd key assets such as authentication servers and mail servers so  
 they can expand the compromise and install remote access Trojans to get a  
 permanent presence. There are many possibilities.

1. In the old days, an attacker would install packet sniffer software to harvest

passwords and compromise other accounts, eventually including a sysad-  
 min’s. Good practice nowadays is to block such attacks using two-factor  
 authentication, or using a protocol such as Kerberos or SSH to ensure that  
 clear text passwords don’t go over the LAN.

2. Other techniques target shared resources such as ﬁle servers. For example,

Linux servers may use the *Network File System* (NFS) protocol; when a  
 volume is ﬁrst mounted, the client gets a *root ﬁlehandle* from the server  
 – an access ticket that doesn’t depend on the time and can’t be revoked.  
 We block this at our own lab using Kerberos to authenticate clients and  
 servers. There are similar problems with Windows ﬁle shares, although the  
 details are different; the EternalBlue vulnerability used by the WannaCry  
 and NotPetya worm exploited such ﬁle shares.

3. Security mechanisms such as SSH bring further vulnerabilities in that

machines in large organisations may have many thousands of SSH keys  
 to communicate with each other, and intruders can exploit them and the  
 trust structures they create to move around.

To get an idea of the range of tools available to a capable attacker nowa-

days, I’d suggest you browse the NSA papers released by Ed Snowden and the  
 CIA toolkits leaked in the Vault 7 disclosure. Cyber warriors have a range of  
 exploit kits, droppers, RATs and software for stealthy exﬁltration of intelligence  
 product.

The takeaway is that the ease with which an intruder on your network can

take over other machines depends on how tightly you have the network locked  
 down, and the damage that can follow any breach will depend on the extent to  
 which other machines in your network trust, or are vulnerable to, the compro-  
 mised machine. This is one of the arguments for not trusting local networks,  
 but insisting on strong authentication between clients and servers at all times.

|  |  |  |
| --- | --- | --- |
|  |  |  |

**21.3.5** **Countermeasures**

Within a few months of the ﬁrst PC viruses appearing in the wild in 1987, there  
 were startups selling antivirus software. This led to an arms race in which virus  
 and antivirus developers tried to outwit each other.

Early antivirus software came in basically two ﬂavours – *scanners* and *check-*

*summers*. Scanners search executable ﬁles for an *indicator of compromise* (IoC),  
 typically a string of bytes from a speciﬁc virus. Malware developers responded  
 in various ways, and the dominant technique became *polymorphism*. The idea is  
 to change the code each time the malware replicates, to make it harder to ﬁnd  
 stable IoCs. The usual technique is to encrypt the code, and have a small header  
 that contains decryption code. With each replication, the malware re-encrypts  
 itself under a different key, and tweaks the decryption code by substituting  
 equivalent sequences of instructions. Modern malware may be run through

half-a-dozen such *packers* in turn, and recursively unpack itself when run. AV  
 ﬁrms ﬁght back by running the code in a virtual machine, so the malware devs  
 include VM-detection code. The AV ﬁrms can at least use the unpacked code  
 as an IoC so long as they can hack through to the last unpacking operation.

Checksummers keep a whitelist list of all the authorised executables on the

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

To provide robust defences against malware, you have to combine tools,

incentives and management. We learned in the old days of DOS-based ﬁle

viruses to provide a central reporting point for all incidents, and to control all  
 software loaded on an organisation’s machines. The main risks were machines  
 used at home both for work and for other things (such as kids playing games),  
 and ﬁles coming in from other organisations. The same principles still apply.  
 However, ﬁrms now need a more coordinated response than before. One of

the reasons is that antivirus software has been getting steadily less effective.  
 The commercialisation of botnets and of machine exploitation has meant that  
 malware writers operate like companies, with research and test departments.  
 Almost all exploits are undetectable by the current antivirus products when ﬁrst  
 launched (if their writers test them properly) and many of them recruit their  
 target number of machines without coming to the attention of the antivirus  
 industry. The net effect was that while antivirus software might have detected  
 almost all of the exploits in circulation in the early 2000s, by 2010 the typical  
 product might detect only a third of them, and by 2020 you expect to detect  
 infection after the fact and have to clear up. That means having good tool  
 support, logging network traffic and analysing it in the light of the latest threat  
 intelligence. What’s more, the rootkit vendors provide after-sales service; if a  
 removal kit is shipped, the rootkit vendor will rapidly ship countermeasures.  
 And nowadays many attackers – especially the competent ones – don’t leave  
 malware ﬁles lying around but ‘live off the land’; they might just add their ssh  
 key to a list of authorised keys on one of your servers so they can pop in when  
 they feel like it, leaving nothing for legacy AV to ﬁnd.

|  |  |  |
| --- | --- | --- |
|  |  |  |

**21.4** **Defense Against Network Attack**

In defending against malware and network attack generally, the view from the  
 second edition of this book in 2008 was that you needed three things: good  
 enough management to keeping your systems patched up-to-date and conﬁgured  
 properly; ﬁrewalls to stop known Trojans and network exploits; and intrusion  
 detection to monitor your networks and machines for indicators of compromise  
 so you can catch the stuff that got through and clean up afterwards.

The principles remain the same in 2020 but reality is much more complex

now, because the scale and complexity of the task have made automation almost  
 essential. A large Windows shop might have something like the following:

1. An agent running on each endpoint, reporting to a cloud service to give

you full visibility of what software is running where and to enable you to  
 push updates;

2. A vulnerability scanner that continually probes your network for known

vulnerabilities;

3. Various boundary control devices which may include ﬁrewalls, a proxy

server that ﬁlters all URLs of websites that staff visit, and proxies for  
 critical applications;

4. An SSL gateway for staff working remotely;

5. A *bring-your-own-device* (BYOD) manager, to control laptops, phones and

other devices that staff members use but that the ﬁrm doesn’t own;

6. A *data leakage prevention* (DLP) system to identify staff who attempt to

remove company documents or code;

7. A threat intelligence platform that integrates feeds from multiple providers,

to alert you to various indicators of compromise including bad DNS names  
 and IP addresses;

8. A log analysis tool that enables you to go back and work out when a

compromise ﬁrst happened, and how far it spread;

9. a *security orchestration and response* (SOAR) system that helps you re-

spond quickly if you note that some devices in your network are commu-  
 nicating with bad addresses such as the command-and-control servers of  
 known malware.

Making all this work together requires system integration, otherwise you’ll

have dozens of staff in your network security centre whose job is to copy lists  
 of bad domains, bad IP addresses and other indicators of compromise from one  
 tool to another.

That said, let’s work our way down this list.

Organisations that are serious about IT security – because they are targets of

state actors (like big service ﬁrms), or have demanding compliance requirements

|  |  |  |
| --- | --- | --- |
|  |  |  |

(like banks), or have a lot to lose (like the military) – aim to stop all vulnera-  
 bilities at source. This means keeping everything patched up to date, which in  
 turn means automated patch management. But such a strategy is harder than  
 it looks. It brings with it a number of hard subproblems, such as maintaining  
 an accurate inventory of all the devices on your network. If you impose a rigid  
 bureaucracy for registering new devices, people will have to ﬁnd ways to cir-  
 cumvent it to get their work done. So you need to also scan your network to  
 see what’s there and whether it’s vulnerable. And even diligent organisations  
 may ﬁnd it’s just too expensive to ﬁx all the security holes at once; patches may  
 break critical applications, and an organisation’s most critical systems often run  
 on the least secure machines, as administrators have not dared to upgrade them  
 for fear of losing service.

This interacts with operational security. In Chapter 2 and Chapter 8 we

discussed the practice and limitations of training staff to not expose systems by  
 foolish actions. By the mid-2000s, the main attack vector was spearphishing –  
 getting people to click on links in email that download and install rootkits. We  
 learned from Ed Snowden that this was the standard way for the NSA to attack  
 a company in 2013: they would monitor external traffic to identify sysadmins, do  
 some background research to identify individual targets, and craft a convincing  
 phishing lure. Alternatively they would direct the target to a website they could  
 spoof or where they could mount a man-in-the-middle protocol attack.

You may try to educate your staff to not click on links in suspicious mail,

but competent attackers create mails that don’t look suspicious. And so many  
 businesses expect their customers and suppliers to click on links that your staff  
 will have to do some clicking to get their work done. We discussed in Chapter 3  
 and elsewhere that victim blaming is maladaptive; if your security systems are  
 not usable, you have to ﬁx them rather than blaming the poor users.

Many ﬁrms mitigate the risk by opening all mail attachments in a cloud

service rather than a local machine, giving staff non-Windows machines such as  
 Chromebooks, iPads or Macs, or having a ﬁrewall or mail ﬁlter that strips out  
 suspicious content.

**21.4.1** **Filtering: ﬁrewalls, censorware and wiretaps**

A *ﬁrewall* is a machine that stands between a private network and the Inter-  
 net, and ﬁlters out traffic that might be harmful. It’s named after the metal  
 bulkhead that separates the passenger compartment of a car or light plane from  
 the engine compartment, to protect the occupants from a fuel ﬁre. Firewalls  
 were controversial when they appeared in the mid-1990s; purists said that all  
 the machines in a company should be secured, while ﬁrewall advocates said this  
 was impractical. The debate has swung back and forth since.

Firewalls are just one example of systems that examine streams of pack-

ets and perform ﬁltering or logging 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 overlap-  
 ping 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. Many ﬁlters also keep logs, so that attacks  
 can be investigated after the fact; and in parts of the ﬁnancial sector, all staff  
 communications are required to be logged so that regulators can investigate any  
 suspicions of insider trading or money laundering.

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.

**21.4.1.1** **Packet ﬁltering**

The simplest kind of ﬁlter merely inspects packet addresses and port numbers.  
 This functionality is available as standard in routers, in Linux and in Windows.  
 You can block IP spooﬁng by ensuring that only ‘local’ packets leave a network,  
 and only ‘foreign’ ones enter. 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 often used to block all traffic except that arriving

on speciﬁc port numbers. You might initially allow the ports used by com-

mon services such as email and web traffic, and then open up further ports as  
 needed. As we move to *software deﬁned networks* (SDN), which replace expen-  
 sive routers with cheap switches controlled by software on commodity servers,  
 packet ﬁltering rules become just the access-control rules in the SDN controller.

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 passes the  
 ﬁrewall’s inspection but is then overwritten by a subsequent fragment, replacing  
 the source address with one that violates your security policy. Another limita-  
 tion is that maintaining a blacklist is difficult, especially 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, phishermen use tricks like  
 fast-ﬂux in which a site’s IP address changes several times an hour.

**21.4.1.2** **Circuit gateways**

The next step up is a *circuit gateway* that reassembles and examines all the  
 packets in each TCP session. This is more expensive than simple packet ﬁltering  
 but can also provide the added functionality of a *virtual private network* (VPN)  
 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, such a ﬁlter can’t screen out bad things at the application level, from  
 malicious code to child sex abuse material. Thus it may be programmed to  
 direct certain types of traffic to application ﬁlters.

|  |  |  |
| --- | --- | --- |
|  |  |  |

**21.4.1.3** **Application proxies**

The third type of ﬁrewall is the *application proxy*, which understands 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 objective is stripping out  
 code, be it straightforward executables, active content in web pages, or macros  
 from incoming Word documents. The move to web-based mail services and the  
 adoption of https have left signiﬁcantly less work for mail ﬁlters to do, and as  
 the service ﬁrms adopt technical measures such as certiﬁcate transparency to  
 prevent proxying, ﬁltering needs to shift to endpoints.

An application proxy can also be a bottleneck. An example is the Great

Firewall of China, which tried through the 2000s to block mail and web content  
 that refers to banned subjects [448]. Since the adoption of https by the major  
 service providers, and the availability of services such as Google Docs that can  
 also be used for communication, China simply stops most of its citizens from  
 using services like Gmail and Facebook.

In the emerging BeyondCorp model promoted by Google, proxies sit in front

of the application servers themselves so that the internal network does not need  
 to be trusted.

**21.4.1.4** **Ingress versus egress ﬁltering**

Most ﬁrewalls look outwards and try to keep bad things out, but some 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 [442]; and by now most consumer ISPs prevent their  
 customers sending packets with spoofed source addresses. This *source address*  
 *validation* means that DDoS operators using UDP reﬂection attacks can no  
 longer use botnets but need to rent servers in data centres.

The fastest-growing use of egress ﬁltering in 2020 is for *data leakage preven-*

*tion* (DLP). Software that ‘phones home’, whether for copyright enforcement or  
 marketing purposes, can disclose highly sensitive material, and prudent organ-  
 isations increasingly wish to monitor and control this kind of traffic. But the  
 pervasive use of https means that DLP systems typically need to install software  
 on endpoints rather than using middleboxes.

**21.4.1.5** **Architecture**

For years, many ﬁrms bought a ﬁrewall to keep their auditors happy. If that’s  
 your pain point, a simple ﬁltering router won’t need much maintenence and  
 won’t get in the way too much. At the other extreme, a serious ﬁrewall system at  
 a defence contractor 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. You may also expect to ﬁnd data diodes separating networks operating  
 at different clearance levels, to ensure that classiﬁed information doesn’t escape

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |
| --- |
| Mail�  proxy  Internet Filter  Filter Intranet  Web�  server Mail�  guard  . . .  Other�  proxies Classified�  intranet |

Figure 21.2: Complex ﬁrewalls for an MLS network

either outwards or downwards (Figure 21.2).

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 some shared central services such as logging.  
 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 from a department of theology. Keeping each network small limits  
 the scope of any compromise and helps incentivise system administrators to  
 defend it.

Considerations in the design of a network security architecture include sim-

plicity, usability, deperimeterisation versus re-perimterisation, underblocking  
 versus overblocking, maintainability, and incentives.

First, since ﬁrewalls do only a small number of things, it’s possible to make

them simple to remove sources of vulnerability and error. If your organisation  
 has a heterogeneous population of machines, then loading as much of the security  
 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. These are roughly the energy utility, and Google, models discussed in the  
 introduction above.

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 the ﬁrewall, to get their work done. I will discuss in  
 section **??** how diplomats have come unstuck by using private email when their  
 official systems were unusable. Many well-run ﬁrms have open guest networks,  
 as does our department; there’s always got to be something that works. And  
 a prudent system administrator will monitor the actual network conﬁguration  
 rather than just relying on ‘policy’.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Third, ﬁrewalls only work until people ﬁnd ways round them. Early ﬁrewalls

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. Then everything moved  
 to Web 2.0 and such ﬁlters became largely ineffective.

Next, there’s deperimeterisation – as Google’s BeyondCorp notes, it’s be-

coming steadily harder to put all the protection at the perimeter, thanks to  
 the the proliferation of phones and 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 infor-  
 mally to advertising-supported web apps. If some parts of your organisation  
 can’t be controlled (e.g. the sales force and the R&D lab) while others must be  
 (the ﬁnance office) then you may need separate architectures. The proliferation  
 of web applications 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. Many ﬁrms tried  
 to block JavaScript in the early 2000s but were beaten by popular web sites that  
 require it. Nowadays it may be impossible to prevent your staff attaching large  
 numbers of IoT devices that just cannot be secured at all [1254].

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 cen-  
 sorship 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.)

And as we’ve repeatedly pointed out, security depends at least as much

on incentives as on technology. A sysadmin who looks after a departmental  
 network used by a hundred people they know, 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.

**21.4.2** **Intrusion Detection**

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*  
 (IDS). 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 [1636]. Examples  
 include:

*•* a machine trying to contact a ‘known bad’ service such as an IRC channel

|  |  |  |
| --- | --- | --- |
|  |  |  |

resolve a known-bad DNS name;

*•* packets with forged source addresses – such as packets that claim to be

*•* spam coming from a machine in your network.

In cases like this, the IDS typically tells the sysadmin that a particular

machine needs to be looked at. This may be just the ﬁrst step in an investigation  
 that involves staring at logs to see how it happened, and what else the attackers  
 might have infected.

Other examples of intrusion detection, which we’ve seen in earlier chapters,

include mechanisms for detecting payment card fraud and stock-market systems  
 that look for insider trading, such as via increases in trading volume just before  
 a price-sensitive announcement. This is now an active area of research: the  
 boom in AI since 2012 has created lots of startups looking for pattern-matching  
 problems.

**21.4.2.1** **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 an account 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 starts to use sophisticated tools like compilers. Simple mis-  
 use detection systems, such as antivirus scanners, look for a *signature* – a known  
 characteristic of a speciﬁc attack. This can be either explicit in the data (such  
 as a substring of an executable ﬁle that marks it as a speciﬁc piece of malware)  
 or in behaviour (such as a machine contacting the IP address of a known botnet  
 command-and-control server). More complex misuse detection systems treat a  
 number of signatures as signals and then train a machine-learning classiﬁer to  
 make the decisions. As I discussed in section 12.5.4, the systems used to detect  
 card fraud use dozens of signals, as they need low false alarm rates to be useful  
 given the scale of modern payment systems.

*Anomaly detection* systems attempt the much harder job of looking for

anomalous 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 have used AI techniques since the 1990s,  
 though some ﬁrms eschew them; Google policy, for example, is to avoid systems  
 that try to learn thresholds or automatically detect causality, and instead have  
 simple systems that detect changes in end-user request rates [236].

The dividing line between misuse and anomaly detection is somewhat blurred.

A 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 when numbers come from ran-  
 dom natural sources and span more than one order of magnitude, 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 ran-  
 dom number generators without knowing Benford’s law, are often caught this  
 way [1247].

Another borderline case is the *honeypot* – 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 ignore patient conﬁdential-  
 ity. In the network context, honeypots emulate many types of device so that  
 attackers scanning the Internet looking for (say) a DSL modem of a particular  
 upgrade status ﬁnd one to attack; this may contain either a simple emulator, or  
 with more recent designs, the actual modem ﬁrmware running in a VM [1955].  
 The upshot is that the honeypot operator gets to see who’s attacking what, and  
 how.

**21.4.2.2** **General limitations of intrusion detection**

Some intrusions are obvious. If you’re worried about activists vandalising your  
 web site, then have a machine somewhere that fetches the page frequently and  
 rings an alarm when it changes. But in the general case, intrusion detection is  
 hard. The virus pioneer Fred 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 [450].

There’s also a matter of deﬁnitions. Some intrusion detection systems are

conﬁgured to block some kinds of suspicious behaviour. But this turns the

intrusion-detection system into an access control mechanism, as well as opening  
 the door to service-denial attacks. I prefer to deﬁne an intrusion-detection sys-  
 tem as one that monitors the logs and draws attention to suspicious occurrences.

Then there’s the cost of false alarms. Academic machine-learning researchers

often consider they’ve done well when they train a classiﬁer to have a false alarm  
 rate of 0.1%. But if you’re on the Gmail team and dealing with a billion users  
 authenticating themselves every day, that’s way too much. Large-scale systems  
 need really low false alarm rates.

Finally, there are three generic problems with machine-learning classiﬁers:

the facts that they’re not much good at detecting new attacks, that people game  
 them, and that they inhale the prejudices of their training data. We will discuss  
 these in more detail in section 25.3.

**21.4.2.3** **Speciﬁc problems detecting network attacks**

Turning now to the speciﬁc problem of detecting network intrusion, it’s harder to  
 spot than payment fraud. Network intrusion detection products still have high  
 missed alarm and false alarm rates. It’s common to detect actual intrusions  
 only afterwards. 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  
 site, and enough of it can be interpreted as hostile to provide a signiﬁcant  
 false alarm rate. 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 [213].

*•* There are ‘too few attacks’. If there are ten real attacks per million sessions  
 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 rate exceeds the disease prevalence. Where  
 the signal is way below the noise, the guards get tired and the genuine  
 alarms get missed.

*•* While a theft from a bank causes an incorrect state – money in the wrong  
 avoid this, for example if their mission is to exﬁltrate conﬁdential data.  
 It’s easier to write software to detect errors than it is to detect slightly  
 odd behaviour.

*•* Many network attacks are speciﬁc to particular versions of software, so  
 However, many ﬁrms buy intrusion detection systems in order to satisfy  
 insurers or auditors, and the products aren’t always kept up to date.

*•* As more and more traffic is encrypted, it can’t easily be subjected to  
 the norm, tools that rely on analysing your DNS traffic will become much  
 less effective.

*•* The issues we discussed in the context of ﬁrewalls largely apply to intrusion  
 a lot; or you can proxy your applications, which is expensive – and needs  
 to be constantly updated to cope with new applications and attacks.

*•* You may have to do intrusion detection both locally and globally. More  
 web sessions; but some attacks are *stealthy* – the opponent sends 1–2  
 packets per day to each of maybe 100,000 hosts, and you need a central  
 monitor that counts packets by source and destination address and by  
 port.

Nowadays, intrusion detection systems involve the coordination of multiple

monitoring mechanisms and products at different levels both in the network and  
 on your ﬂeet of endpoint devices. A large company with tens of thousands of  
 staff using Windows will typically have several dozen products, as I discussed  
 previously in section 21.4. Integrating and automating both monitoring and

|  |  |  |
| --- | --- | --- |
|  |  |  |

response makes up more and more of a CISO’s job. The growth areas therefore  
 include integration tools for *security incident and event management* (SIEM),  
 *security orchestration and response* (SOAR), and metrics.

**21.5** **Cryptography: the ragged boundary**

Network security interacts with cryptography in a number of ways. We already  
 mentioned the debate about DNS over https; now I’m going to describe ﬁve  
 other aspects of crypto brieﬂy. They are SSH; the local link protection offered  
 by WiFi, Bluetooth and HomePlug; the IPSec mechanisms used in VPNs; TLS;  
 and the *public key infrastructures* (PKI) used to support many of these. In

the previous chapter, we discussed how attempts to build more trustworthy  
 components out of cryptography run up against many real-world engineering  
 and economic constraints. The tools that we use to set boundaries on networks,  
 and to translate trust within them, are no different.

The emerging themes are that the most distributed part of the problem is

unmanageable because the vendors don’t care; in particular the thousands of  
 device types being marketed as part of the ‘Internet of Things’ have no remote  
 management facility available to users, the vendor often doesn’t upgrade the  
 software, and the lack of a user interface means that authentication is haphazard  
 at best. Meanwhile the most centralised part of the problem – PKI – is often  
 subverted by government mandates.

**21.5.1** **SSH**

When I use my laptop to access ﬁles on my desktop machine, or do anything  
 with any other machine in our lab for that matter, I use *secure shell* (SSH)  
 which provides encrypted links between Unix and Windows hosts. So when I  
 work from home, 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 Yl¨onen, a researcher at Helsinki

University of Technology, following a password-sniffing attack there [2058]. It  
 sets up encrypted connections between machines, so that logon passwords don’t  
 travel across the network in the clear, and supports other useful features that  
 led to its rapid adoption [1617].

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 my laptop  
 to a server at the lab, I install my laptop public key in a ﬁle on the relevant  
 server. When I wish to log on to a server I’m prompted for my passphrase;  
 the two machines set up a Diffie-Hellman key; the private keys are used to sign  
 the transient public keys, to stop middleperson attacks; the subsequent traffic  
 is thus both encrypted and authenticated. Manual key installation is intuitive,  
 but doesn’t scale particularly well. There are also options to use Kerberos,

whether to authenticate the session key set up using Diffie-Hellman, or to set

|  |  |  |
| --- | --- | --- |
|  |  |  |

up the session key directly. (In the latter case, SSH falls back to being a variant  
 of Kerberos in the sense that it is now a trusted third-party protocol, and the  
 police can get the Kerberos server to decrypt the traffic.)

Possible problems include the fact that if you’re typing at the keyboard

one character at a time, then each character gets sent in its own packet, and  
 the packet interarrival times can leak a lot of information about what you’re  
 typing [1803]. However, the worst is probably that most SSH keys used for

server-to-server communication are stored in the clear, without being protected  
 by a password at all. So if a server is compromised, the same can happen to  
 every other machine that trusts an SSH key installed on it.

SSH is often used as a simple logon mechanism; many IoT devices run Linux

and allow remote logon by anyone who knows an appropriate password. This  
 opens them to password-guessing attacks, and where there are weak passwords  
 or a known default password, to recruitment into botnets based on Mirai and  
 similar tools. The countermeasure here is honeypots.

**21.5.2** **Wireless networking at the periphery**

Many networks use wireless technology at the edge to go the last few feet from  
 an access point to a device, or from one device to another. Protocols such as  
 WiFi, Bluetooth and Homeplug all offer encryption to provide some protection  
 against service abuse and perhaps against eavesdropping. However most are  
 vulnerable to local attacks that are difficult to block completely because many  
 devices don’t get patched, lack user interfaces, or both.

**21.5.2.1** **WiFi**

WiFi supports wireless local area networks, whether at home to connect phones  
 and other devices to a home router, or by businesses to connect payment ter-  
 minals and stock control devices as well as PCs. It 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 be-  
 cause of the weak ciphers demanded by US export control and poor protocol  
 design [299, 1873]. Since 2004, an improved system called WPA2 uses AES en-  
 cryption. The key for each access point is typically printed on a card that ﬁts  
 into the back of the router.

Should WiFi networks be seen as untrusted? The reason to set a password

is more to prevent third parties using your bandwidth or quota, rather than the  
 risk of pharming. Many people in the UK or America ﬁnd it convenient to have  
 an open network for guests to use, and so that you and your neighbours can  
 use each others’ networks as backups. In countries where you pay for download  
 bandwidth, home router passwords are mostly set. In some, like India, it’s

against the law to run an open WiFi access point (terrorists who mounted an  
 attack in Bombay in 2008 used them to call home unobtrusively). Having the  
 key on a card is a neat example of usable security design: the householder can  
 make their network as open or as secure as needed by pinning the card on the  
 wall or by locking it up.

|  |  |  |
| --- | --- | --- |
|  |  |  |

WiFi security is still somewhat fragile. *Universal Plug and Play* (UPnP) lets

any device in a network punch a hole through the router’s ﬁrewall; DHS has  
 been recommending since 2013 that people turn it off. However now that many  
 devices and domestic appliances come with an attached cloud service, that’s  
 hard. It’s used along with *WiFi Protected Setup* (WPS) which lets you enrol  
 gadgets on your network with a simple button press. You can set a PIN but  
 there have been a couple of attacks found on the mechanism.

Businesses may have to take a bit more care. In March 2007, retail chain TJ

Maxx reported that some 45.7 million credit card numbers had been stolen from  
 its systems; the Wall Street Journal reported that an insecure WiFi connection  
 in St Paul, Mn., was to blame [1509]. Banks sued the company, and eventually  
 settled for $41m [788].

Patching is an issue. For example, in March 2020 we learned of the Kr00k

vulnerability in Broadcom wiﬁ chips which will get patched in Macs and iPhones  
 but probably not in wireless routers or older Android phones [799]. As for the  
 great majority of IoT devices, from toys through home appliances, they won’t  
 get patched, ever.

**21.5.2.2** **Bluetooth**

Bluetooth is another short-range wireless protocol, aimed at *personal area net-*  
 *works*, such as linking a headset to a phone, or a phone in your pocket to a  
 hands-free interface in your car. It’s also used to connect cameras and phones  
 to laptops, keyboards to PCs and so on. Like WiFi, the ﬁrst versions of the  
 protocol turned out to have ﬂaws [2015, 1713, 1101]. From version 2.1 (released  
 in 2007), Bluetooth has supported Secure Simple Pairing [1169], which uses  
 elliptic-curve Diffie-Hellman to thwart passive eavesdropping attacks. Man-in-  
 the-middle attacks are harder; they are dealt with by generating a six-digit  
 number for numerical comparison. However, because one or both of the devices  
 might lack a keyboard or screen (or both), it’s also possible for the number to  
 be generated at one device and entered as a passkey at another; and there’s  
 a ‘just works’ mode that’s not protected against middleperson attack. What’s  
 more, the data may or may not be signed, giving a total of about ten different  
 combinations of conﬁdentiality, integrity and resistance to man-in-the-middle  
 attack; and a number of attacks have been found, some inspired by NSA tools  
 listed in the Snowden disclosures [1635]. Again, patching is an issue. In 2018,  
 Eli Biham found that many implementations could be fooled by a man-in-the-  
 middle supplying an invalid elliptic curve to the authentication protocol [244],  
 and in 2020 Daniele Antonioli and colleagues discovered a variant of the mig-in-  
 the-middle attack where you just reﬂect the challenge from a bluetooth device  
 back to it, claiming that you’re now the challenger and the target device is the  
 responder [124]. So if you have a device with a bluetooth chip that hasn’t been  
 patched, it may be vulnerable.

**21.5.2.3** **HomePlug**

HomePlug is a protocol used for communication over the mains power cables.  
 HomePlug AV is widely used in wiﬁ extenders: you plug one station into your

|  |  |  |
| --- | --- | --- |
|  |  |  |

router or cable modem, and another gives a remote wiﬁ access point at the  
 other end of your house. (Declaration of interest: I was one of the protocol’s  
 designers.) 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. We decided to offer only two modes of operation: secure mode, in which  
 the user manually enters into their network controller a unique AES key that’s  
 printed on the device label, and ‘simple connect’ mode in which the keys are  
 exchanged without authentication. The keys aren’t even encrypted in this mode;  
 its purpose is not to provide security but to prevent wrong associations, such as  
 when a device wrongly mates with a network next door [1436]. However many  
 vendors just support the ‘simple connect’ mode and end up with a policy of trust  
 on ﬁrst use, as already mentioned in section 14.3.3.3. Others sell extenders

in pairs, with keys already installed. There are variants for smart meters to  
 communicate with substations, and for electricity utilities to provide broadband  
 to the home over the power line (though these are not widely used because of  
 radio frequency interference). Vendors also customised the product in various  
 ways to make it incompatible with competitors. As a result of this mess, little  
 reliance can be placed on the key management.

**21.5.2.4** **VPNs**

*Virtual private networks* (VPNs) typically do encryption and authentication at  
 the IP layer using a protocol suite known as IPsec. This deﬁnes a *security asso-*  
 *ciation* as the combination of keys, algorithms and parameters used to protect  
 a particular packet stream. Protected packets may be just authenticated, or en-  
 crypted too; in the former case, an authentication header is added that protects  
 data integrity, while in the latter the packet is also encrypted and encapsulated  
 in other packets. There’s also an *Internet Key Exchange* (IKE) protocol to set  
 up keys and negotiate parameters, and we may infer from Ed Snowden’s disclo-  
 sures that the standard default settings of this (with 1024-bit Diffie-Hellman)  
 are insecure.

VPNs are offered by ﬁrewall vendors so that 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 workers’ laptops and home  
 PCs can also join a VPN given appropriate software. VPNs are also offered  
 commercially, and are used for example by people and ﬁrms in countries like  
 Iran and China to circumvent the national ﬁrewall.

**21.6** **CAs and PKI**

As we discussed in section 5.7.4, the pioneers of public-key cryptography devel-  
 oped a vision of certiﬁcates that would bind public keys to the names or roles  
 of the organisations, people or devices that controlled the corresponding private  
 keys. Initially it was thought that governments or phone companies would do  
 this, but they were too slow. During the dotcom boom, entrepreneurs set up  
 *certiﬁcate authorities* (CAs) and software ﬁrms such as Microsoft and Netscape  
 embedded their public keys into their browsers. There followed a gold rush as

|  |  |  |
| --- | --- | --- |
|  |  |  |

the CAs bought each other and consolidated; investors hoped that every device  
 would need a public-key certiﬁcate, so you’d need to pay Verisign ten bucks  
 every two years to renew the certiﬁcate on your toaster, or it wouldn’t talk to  
 your fridge.

Once that foolishness died down, the world’s governments moved to get their

own CAs’ root certiﬁcates into the browsers for intelligence and surveillance pur-  
 poses. As people moved to web services like Gmail, security agencies developed  
 tools to do man-in-the-middle attacks, and as TLS was used to encrypt pass-  
 word entry (and later, the whole session), this meant having a CA that would  
 produce a certiﬁcate on www.gmail.com for a security agency public key that  
 the target’s browser would accept. In fact, at a panel discussion at Financial  
 Cryptography 2011, I asked the man from Mozilla how come, when I updated  
 Firefox the previous day, it had put back a certiﬁcate I’d removed for Tubitak –  
 a Turkish intelligence organisation. At this point a man stood up in the audience  
 and shouted ‘How dare you insult my country! Tubitak is not an intelligence  
 agency – it is a research organisation!’ The man from Mozilla shrugged and said  
 wryly, ‘Now you see how hard certiﬁcate governance is.’

Later that year came the DigiNotar scandal. DigiNotar was a Dutch CA

which was found to have issued wildcard certiﬁcates for Gmail. Iranian agents  
 had hacked it in order to monitor 300,000 Gmail users in Iran; sanctions meant  
 that, unlike Turkey, they could not just have their government certiﬁcate in-  
 stalled in the major browsers. Mozilla and Google promptly put DigiNotar to  
 death by removing its root certiﬁcates; Microsoft and Apple followed quickly.  
 This caused real disruption in the Netherlands, many of whose online govern-  
 ment services used DigiNotar certiﬁcates, and had to scramble to get others.  
 It turned out that there had been earlier attacks on another CA, Comodo, but  
 that company claimed to have revoked all its wrongly-issued certiﬁcates. Since  
 then, there has been increasing pressure on CAs and auditors from the browsers’  
 root stores.

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*. If you’re building a service that government  
 agencies are likely to attack, then it may be a good idea to keep your PKI  
 closed, with a CA that runs on your own premises – so you get to know of any  
 warrants. I advise ﬁrms who maintain software that’s installed on many millions  
 of machines to use a private CA for their code signing keys.

PKI has a number of intrinsic limitations, many of which we discussed in the

chapter on distributed systems. Naming is difficult, and the more applications  
 rely on a certiﬁcate, the shorter its useful life will be. You can sometimes

simplify things by removing unnecessary names: rather than one certiﬁcate  
 saying ‘Ross Anderson’s key is KR’ and another saying ‘Ross Anderson has the  
 right to administer x.foo.com’ you might just say ‘KR has the right to administer  
 x.foo.com.’

This is an aspect of the ‘one key or many’ debate. Should I expect to have

a single digital credential 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 credential? Multiple keys protect the customer:  
 I don’t want to have to use a key with which I can remortgage my house to buy  
 my lunchtime sandwich. As we saw in the chapter on banking and bookkeeping,  
 it’s easy to dupe people into signing a message by having the equipment display  
 another one.

Now the standard PKI machinery (the X.509 protocol suite) was developed

to provide an electronic replacement for the telephone book, so it started off by  
 assuming that everyone will have a unique name and a unique key in an open  
 PKI architecture.

This in turn leads to issues of trust, of which there are many.

*•* If you remove one of the hundreds of root certiﬁcates from Firefox, then  
 cates – but you can’t delete them at all. In each case, you have to know  
 how to mark a certiﬁcate as untrusted.

*•* There have been some interesting effects where a government that had its  
 military coup in 2014) had to resort to different surveillance methods for  
 Mac users [1554].

*•* Many ﬁrms use certs that are out-of-date, or that correspond to the wrong  
 to run some promotion or another. As a result, users have been trained to  
 ignore security warnings, and only a small minority used to pay attention  
 to them [841]. Recently browsers such as Firefox have made it harder to  
 click past warnings.

*•* Certs bind a company name to a DNS name, but their vendors are usually  
 the applicant can answer an email sent to that domain, or put up a web  
 page with a CA challenge on it. Things are slightly better with ‘extended  
 validation’ certiﬁcates6, but even they aren’t foolproof.

*•* On their ‘certiﬁcation practice statements’ CAs go out of their way to

*•* Certiﬁcate revocation is an issue. The original idea was that anyone rely-  
 CA and check any cert on which they were about to rely. However, this  
 vitiated much of the beneﬁt of public-key cryptography by requiring online  
 operation for high assurance. In addition, users of some systems (partic-  
 ularly US government ones) had to download large CRLs every time they  
 started up their systems, leading to delay and network congestion. Since  
 about 2013, people have moved to the *Online Certiﬁcate Status Protocol*  
 (OCSP), a more efficient protocol for online status checking.

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

6These used to bring up a green padlock in your browser, though this is being discontinued

in Chromium from v 76 in 2020 after research showed that nobody paid any attention.

|  |  |  |
| --- | --- | --- |
|  |  |  |

complex and heavyweight protocol called SET, because it placed less of a burden  
 on developers [110]. The costs of compliance were dumped on the users – who  
 are often unable to cope [524]. Much of the engineering around CAs and certs  
 since then has been playing catchup.

The big issues at the time of writing are certiﬁcate lifetime; LetsEncrypt;

and certiﬁcate transparency.

The maximum permitted lifetime of a certiﬁcate, if it’s to be accepted by

the main browsers, has steadily reduced from 8 years to 3 years to 27 months.  
 Ballots in 2017 and 2019 proposed a cut to 13 months [1581] and in 2020 Apple  
 forced the issue by declaring that from September, its devices would no longer  
 accept any certs valid for longer than 398 days [1446]. This will force many  
 websites to refresh their certiﬁcates; it will be interesting to see how ﬁrms ﬂush  
 out all the certs in DNS. (It will also widen the gap between systems with  
 annual certs and some industrial and IoT systems where certs have to last for  
 years because of the difficulty of software upgrade.)

Getting certs used to be difficult as you had to go shopping for one, prove

you controlled your domain, get the cert, upload it to your server, change the  
 conﬁguration and then test it all. The change maker here has been a nonproﬁt,  
 the Internet Security Research Group (ISRG) which provides certs for free and  
 by February 2020 had issued a billion of them. Making certs free allowed full  
 automation, which keeps costs down: their ‘LetsEncrypt’ CA supports 100m  
 sites on a budget of $3m pa. LetsEncrypt set out to make deploying certs easy,  
 and the impact has been real: 20% of browser connections are still in plaintext,  
 but this is down from 60% four years ago. This service started in 2015, two  
 years after the Snowden revelations. Their automated certiﬁcate management  
 environment is now standardised as RFC8555, so commercial CAs are using it  
 too. There’s a transparency log and the system has no manual override, so

there’s some assurance that they have never been compelled to issue a cert. (In  
 fact, the NSA uses their certs.) At November 2019, they were the largest CA,  
 with 112m certs for 188m domains; they had 5% of the top hundred sites but  
 35% of the top million. Their scale means that mistakes affect lots of sites; in  
 March 2020, a bug in their software meant that 3 million certiﬁcates covering  
 12 million server names had to be replaced [590].

**21.6.1** **Certiﬁcate transparency**

Following the attacks on Comodo and DigiNotar, work started on mechanisms  
 to block maliciously issued certiﬁcates. Certiﬁcate transparency sets out to do  
 this by maintaining logs of all the certiﬁcates seen in the wild for each domain,  
 so that domain owners can rapidly spot certs that should not have been issued  
 for their domain. Google launched the ﬁrst certiﬁcate transparency log in 2013  
 and Chrome started insisting on such logs for extended validation certiﬁcates  
 in 2015. Google found that Symantec had issued certiﬁcates for a number of  
 domains (including their own) without the domain owner’s knowledge [1786],  
 and made certiﬁcate transparency mandatory for all CAs in 2018.

|  |  |  |
| --- | --- | --- |
|  |  |  |

**21.7** **Topology**

The topology of a network is the pattern in which its nodes are connected, and  
 this can be a signiﬁcant component of the security architecture.

*•* A utility might have a number of islands, each containing a generator  
 connected in turn via a specialised ﬁrewall and a VPN to a network control  
 centre.

*•* A cloud service provider might have tens of thousands of machines in a  
 and its tenants determining which VMs or containers on which machines  
 can communicate with each other. And while the internal network may be  
 untrusted, in the sense that network location plays no role in access control  
 decisions, it may be shielded from DDoS attacks by front-end systems.

*•* Classiﬁed systems used by governments may have quite large trusted net-

More complex topologies can be found where nodes are users and edges are

their presence in each others’ address books. Social-network analysis has been  
 applied to disciplines from epidemiology through criminology and the study of  
 how new technologies diffuse, to the study of harms transmitted directly between  
 users, such as macro viruses [1433]. Social networks can be modelled by a graph  
 with a power-law distribution of vertex order; a small number of well-connected  
 nodes help make the network resilient against random failure, and easy to navi-  
 gate. Yet they also make such networks vulnerable to targeted attack. Remove  
 the well-connected nodes, and the network is easily disconnected [36]. Dictators  
 have known this intuitively; Stalin consolidated his rule by killing the richer  
 peasants, Pol Pot killed intellectuals, while William the Conqueror killed the  
 Saxon gentry. Now we have quantitative models, they help explain why revo-  
 lutionaries have tended to organise themselves in cells [1373]; 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 – unless the dissidents  
 organised in a cell structure in the ﬁrst place [510].

**21.8** **Summary**

Preventing and detecting attacks that are launched over networks is the core of  
 a modern CISO’s job. It’s difficult because it involves a huge range of attack  
 types and security technologies. It can lead to newsworthy failures. There is  
 unlikely to be any magic solution, though a lot of things can help. Each new  
 advance opens up new things to worry about; for example, cloud services may  
 shift much of the network security task to a provider, but make conﬁguration  
 management more critical. Overall, the problems are so complex and messy  
 that managing them needs a whole-system approach with automation.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Hacking techniques depend partly on the opportunistic exploitation of vul-

nerabilities introduced accidentally by the major vendors, and partly on tech-  
 niques to social-engineer people into running untrustworthy code. However these  
 have developed into a whole ecosystem of bad guys, which a security engineer  
 also needs to study and understand.

**Research Problems**

In 2000, 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. By 2010, there was much more  
 discussion of economics and policy: of how changing liability rules might make  
 things better [97]. By 2020, there is much more work on metrics: on measuring  
 the actual wickedness that goes on, and feeding this not just into the policy  
 debate but also into law enforcement. At the operational level, the game is  
 about automation and integration – about enabling large ﬁrms to process large  
 quantities of threat intelligence and network surveillance information, turn it  
 into actionable intelligence, and measure how effectively the network security  
 team is doing its job.

**Further Reading**

The early classic on Internet security was written by Steve Bellovin and Bill  
 Cheswick, with Avi Rubin joining them for the second edition [221]. The seminal  
 work on viruses is by Fred Cohen [450], while Java security is discussed by Li  
 Gong (who designed it) [783]. For BGP security, see our 2011 ENISA report:  
 the full Monty is over two hundred pages, designed for people starting a PhD  
 in network security, but there’s a shorter executive summary too [1906].

For a more detailed overview of malware, I might suggest Wenke Lee’s Cybok

survey paper [1137]; and Sanjah Jha’s Cybok survey of network security provides  
 more detail of IPSEC as well as ethernet and port-based security [983].

I’m not aware of any good overview of the certiﬁcation authority ecosys-

tem. You might start with the 2004 oral history interview with Jim Bidzos,  
 the founder of Verisign [240]. The initial goal of Microsoft and Netscape was  
 to jump-start electronic commerce on the worldwide web; certiﬁcate use then  
 spread to passwords and software updates, and when Javascript came along,  
 the same origin principle shifted trust to websites. Many other players jumped  
 in, with some government agencies trying to undermine the CA ecosystem and  
 others trying to reinforce it. There’s conﬂict between technical security goals  
 and legal goals, as well as between auditors and regulators. So there are quite  
 separate views on CA security from WebTrust (the American and Canadian ac-  
 countants) and ETSI (the most relevant European standards body). For more  
 detail, a presentation by Ryan Sleevi on what’s wrong with the ecosystem [1785]  
 has many pointers for those who want to dig into the current problems, both  
 technical and operational, and their background.

|  |  |  |
| --- | --- | --- |
|  |  |  |