Chapter 10

Firewalls and Tunnels

The official version of this book is available at
https://www.springer.com/gp/book/9783030834104

ISBN: 978-3-030-83410-4 (hardcopy), 978-3-030-83411-1 (eBook)

version: 15 Oct 2021

Chapter 10

Firewalls and Tunnels

This chapter discusses perimeter-based defenses, starting with firewalls and then complementary enabling technologies for securing network communications of remote users and distance-separated peers. Generic tools called *encrypted tunnels* and *virtual private networks* (VPNs) are illustrated by SSH and IPsec. We consider risks of network-accessible services and how to securely provide such services, building familiarity with network defense options (and their limitations). Many examples put security design principles into practice, and give reminders of the primary goals of computer security: protecting data and passwords in transit, protecting resources from unauthorized network access and use, and preserving the integrity and availability of hosts in the face of network-based threats.

As a simplified view, firewalls at enterprise perimeters keep out the bulk of unauthorized traffic; intrusion detection systems provide awareness of, and opportunities to ameliorate, what gets through; user traffic is cryptographically protected by technologies such as IPsec-based VPNs, SSH, TLS, and encrypted email; and authentication of incoming packets or connections is used to distinguish authorized entities and data.

This view helps convey an important message: the rich flexibility and functionality enabled by network-accessible services come with security implications. Remote access to network-based services should be over cryptographically secured channels, complemented by mechanisms that allow monitoring of traffic, and at least partial control of where it may flow. As an upside, encrypted network communications provide legitimate parties protection for transmitted data including passwords, and remote access to trusted environments; as a downside, when intruders or malicious insiders use the same tools, the content of their communications is inaccessible, heightening the importance of proper access control and authentication, policy enforcement at entry and exit points, and monitoring-based intrusion detection.

10.1 Packet-filter firewalls

A network security *firewall* is a gateway providing access control functionality that can allow or deny, and optionally modify, data passing between two networks, or a net-

work and a device. The design intent is that traffic cannot bypass the firewall in either direction—thus in theory, packets undergo COMPLETE-MEDIATION (P4). The terminology reflects fire-resistant doors designed to isolate damage and contain spread in case of fire, in line with principle P5 (ISOLATED-COMPARTMENTS). Network firewalls most commonly serve in *perimeter-based defenses*, protecting a trusted private (internal) network from an untrusted public (external) network, e.g., the Internet.

INBOUND AND OUTBOUND. From the private network viewpoint (Figure 10.1), packets arriving are *inbound*, and those leaving are *outbound*. Filtering inbound packets protects the internal network from the Internet. Filtering outbound packets allows awareness and partial control of data sent out and services accessed, e.g., to enforce a security policy restricting allowed protocols and services, and to detect unauthorized transfers (*data extrusion* or *exfiltration*) from compromised internal machines or *insiders*—rogue employees or individuals abusing resources from within. We discuss firewalls under two broad headings: *packet filters* (below), and *proxy-type firewalls* (Section 10.2).

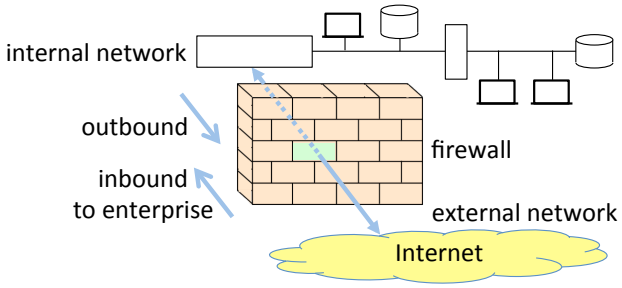

Figure 10.1: Network firewall (basic model).

PACKET-FILTER RULES AND ACTIONS. A packet-filter firewall is configured by an administrator. It contains a list of rules of the form <condition, action>. In a "first-matching rule" firewall, the action taken for a packet is that specified by the first rule whose condition it satisfies. The primary actions are:

- ALLOW (permit packet to pass);
- DROP (silently discard the packet—a type-1 deny); or
- REJECT (drop but also try to inform the source—a type-2 deny). This might result in sending a TCP RST (reset) packet, or for UDP an ICMP "destination unreachable".

In addition to one of the above, a second action may log the packet, e.g., using the `syslog` general system-logging service. For efficiency, most packet-filter matching rules are based on five TCP/IP header fields (`src_addr, src_port, dst_addr, dst_port, prot`),[1] and if ICMP then ICMP type and code. Other header fields (packet size, flags) are sometimes used. More complex rules, and so-called *intelligent packet filtering*, may involve payload data, e.g., an allow or deny decision based on a payload URL—but examining application payloads is generally beyond the scope of packet filters.

[1]Shorthand here is as follows: *src* (source), *dst* (destination), *addr* (address), *prot* (IP header protocol, e.g., TCP, UDP or ICMP). Section 10.6 background on basic networking concepts is assumed to be known.

STATELESS AND STATEFUL FILTERS. In a simple *stateless packet filter*, each packet is processed independently of others (with no dependency on prior packets). In contrast, a *stateful packet filter* keeps track of selected details as packets are processed, for use in processing later packets. State details are kept in a firewall *state table*. This often means tracking TCP connection states; packets with sources corresponding to established or in-progress connection set-ups are treated differently than new sources. Firewalls that track connection-related socket details (i.e., IP addresses, ports) may be called *dynamic packet filters*; this term is also used more generally for a stateful packet filter whose rules automatically change "on the fly" in specific contexts (FTP example, page 286).

Example *(Packet-filtering rules).* Table 10.1 gives sample filtering rules in six blocks.

1. (ingress/egress) Rules 1–2 mitigate spoofed source IP addresses. Discussion of these rules is best deferred until Section 11.3.
2. (SMTP email) Rule 3 denies packets from a known spam server; 4 and 5 allow incoming connections to a gateway mailserver and responses from it; 6 and 7 allow, from the gateway mailserver, outgoing mail connections and associated responses.
3. (HTTP) Rules 8–A allow outbound HTTP connection requests, and inbound responses, but reject inbound HTTP connection requests. Checking for presence of the ACK flag, as listed, stops unsolicited inbound packets; TCP stacks themselves will reject packets with an ACK flag but no associated existing connection.
4. (DNS): Rule B allows outgoing queries from a gateway DNS server, to resolve addresses for internal users; C and D allow incoming queries and related responses, e.g., for authoritative answers to external parties resolving addresses for their users.
5. (ICMP ping): Rules E–H are best discussed with denial of service (Chapter 11).
6. (default deny): Rule Z is last. A packet matching no other rules is blocked.

A (stateful-filtering) rule might ALLOW inbound SYN-ACK packets from external src_addr (s) to an internal dst_addr (d) only if the state table shows d recently sent s a SYN packet.

DEFAULT-DENY RULESETS. Chapter 1's principle P2 (SAFE-DEFAULTS) motivates default-deny firewall rulesets, with a packet allowed only on explicitly matching an accept rule. Such rulesets are constructed from security policies explicitly stating allowed types of access. A default-allow alternative that allows any packet for which no rule explicitly blocks it, has tempting usability (disrupting fewer external services desired by internal hosts)—but is unnecessarily dangerous, as a firewall administrator cannot possibly know of all exploits that might arise or already be in use, all requiring explicit deny rules.

FIREWALLS AND SECURITY POLICY. A packet filter executes precise rules determining which packets may enter/exit an enterprise network. Thus a firewall instantiates, in part, an organization's Internet security policy.[2] If server ports were guaranteed to be bound to known services, then by creating ALLOW rules that translate authorized external services to those ports, outbound service requests could be filtered based on destination ports in TCP headers. In practice this fails, as the mapping between port and service is

[2]To put this in context with broader security policy, firewall rules specify allowed Internet services, e.g., in terms of remote access, whereas file permissions typically relate to local or internal files.

Rule #, Action		Path	Source		Destination		Protocol	Extra field	Comments
			addr	port	addr	port			
1	NO	in	us	*	*	*	*		ingress and egress
2	NO	out	them	*	*	*	*		filtering (Sect. 11.3)
3	NO	in	listed	*	*	*	*		block bad servers
4	OK	in	them	high	GW	25	TCP		inbound mail...
5	OK	out	GW	25	them	high	TCP	ACK	...our responses out
6	OK	out	GW	high	them	25	TCP		SMTP mail out...
7	OK	in	them	25	GW	high	TCP	ACK	...inbound response
8	OK	out	us	high	them	80	TCP		HTTP request out...
9	OK	in	them	80	us	high	TCP	ACK	...allow responses
A	NO	in	them	*	us	80	TCP		no inbound requests
B	OK	out	GW	53	them	53	UDP		our DNS queries
C	OK	in	them	*	GW	53	UDP		DNS queries to us...
D	OK	out	GW	53	them	*	UDP		...responses from us
E	OK	in	them	–	us	–	ICMP	8	pings to us...
F	OK	out	us	–	them	–	ICMP	0	...our responses
G	OK	out	us	–	them	–	ICMP	8	our pings out...
H	OK	in	them	–	us	–	ICMP	0	...responses to us
Z	NO	*	*	*	*	*	*		default deny

Table 10.1: Packet-filtering rule examples (illustrative, for discussion). Notation: NO (deny packet), OK (allow packet), in/out (inbound/outbound packet direction), us/them (internal/external addresses; products specify explicit ranges), * (any value matches), high (port above 1023, unprivileged), listed (list of bad addresses, e.g., spam servers), ACK (ACK bit set), GW (our enterprise gateway mail server/DNS server). For ICMP rules, values (8, 0) refer to ICMP message types (page 305); recall that ICMP does not use ports.

only as trustworthy as the service's host; allowing connections to an untrustworthy host is always dangerous. Nonetheless, outbound HTTP connections (to port 80) are typically allowed as a security-usability tradeoff (possibly with some sites badlisted); and similarly for HTTPS (port 443). Inbound connections from unknown hosts are commonly blocked, as a primary function of firewalls, but firewall rules may authorize (by allowlist) connection request packets from certain IP addresses. In cases where incoming connections are not entirely blocked, one approach is use of firewall rules that allow packets to a small number of known sockets corresponding to internal hosts running authorized services.[3] If these internal hosts are maintained as trustworthy machines, then external clients should have access only to authorized APIs—presumably limiting internal exposure (although exploitable vulnerabilities in such services may remain).

FIREWALLS AS CHOKEPOINTS. Firewalls use the entry point to a private network as a central point for monitoring, control, and packet rejection. The idea is to force traffic through a narrow passageway or *chokepoint*. This implicitly assumes a perimeter, with alternate entry points no easier to get through—like a ticket-check at a sports arena, or an agreed border crossing point between two hostile nations. As noted in Chapter 9,

[3]This is commonly implemented using a so-called DMZ as discussed later in Figure 10.4.

employees originally accessed the web through enterprise "world wide web" *gateways* or proxy servers; these evolved into firewalls. While firewalls remain in wide use, the perimeter model can no longer fully control contact to an internal network, as perimeters with single-point network access have largely disappeared—internal network hosts now commonly access web content (software, data, connectivity) while bypassing firewalls, e.g., via wireless access points not under enterprise management, USB flash drives inserted into hosts, and users in bring-your-own-device (BYOD) environments connecting smartphones directly into internal networks. Firewalls nonetheless remain useful:

- to protect legacy applications within contained subnetworks;
- as an enforcement point for Internet security policy, to monitor and control incoming access by remote adversaries lacking wireless or physical access to hosts; and
- to instantiate accepted defensive principles like defense-in-depth and isolation.

LIMITATIONS. While firewalls play an important contributing role and remain a primary defense line and gateway to the Internet, they have recognized limitations:

1. *topological limitations*: firewall protection assumes true perimeters exist.
2. *malicious insiders*: users and hosts inside a firewall are treated as trusted, implying little protection from users cooperating with outsiders, or intruders with established positions as insiders. Traditional firewalls alone are not intruder-detection systems.
3. *trusted users making bad connections*: firewalls provide little protection when trusted users originate connections resulting in malicious active content or drive-by downloads (Chapter 7) from bad or compromised web sites.
4. *firewall transit by tunneling*: firewall rules relying on port number (to allow only permitted services) are commonly bypassed,[4] by tunneling one protocol through another (Section 10.4). Such tunneling is used for both benign and malicious purposes.
5. *encrypted content*: *content-based inspection* at firewalls (albeit not a basic packet-filter function) is precluded by encryption, unless means are provided to intercept and decrypt connections via proxy-type functionality (again, Section 10.2).

‡**Exercise** (Pros and cons of packet filtering). Summarize the advantages and disadvantages of packet-filtering firewalls (hint: [41, p.108–109]).

‡**Example** *(Dynamic packet filtering: FTP normal mode).* In FTP file transfer, one TCP connection is used as a *control channel* (for commands), and another as a *data channel* (for data transfer). TCP ports 21 (FTP command) and 20 (FTP data) are respectively reserved at the server end. To initiate a "normal mode" FTP session, the client assigns two ports above 1023—one each for command and data—and using its own command port and server port 21, opens a TCP connection on which it then sends an FTP `PORT` command. This tells the server which client port to use for data transfer. The server then opens a TCP connection to the client's data port. This, however, violates the rule of thumb of allowing outbound but refusing externally originated TCP connections. (Note:

[4] This assumes that the firewall lacks capabilities of *content inspection* and application verification, which may be provided by (and motivate) proxy firewalls as discussed later in Section 10.2.

FTP also has a "passive" mode that avoids this issue.) This issue may be avoided by tracking outbound FTP `PORT` commands and the port number that a specific internal host (IP address) has requested be used for inbound FTP data connections, and automatically creating a dynamic (temporary) rule allowing the out-to-in connection to that port. *Dynamic packet filters* may do so; this term may also imply a proxy-type filter (Section 10.2) that is stateful.

‡**Exercise** (Network address translation). Explain what *network address translation* (NAT) is, and its relationship to network security (hint: references in Section 10.7).

‡**DEDICATED FIREWALLS AND HYBRID APPLIANCES.** Firewalls are instantiated as a collection of one or more software and/or hardware components. Commercial routers commonly have packet-filtering capabilities, and may be called *screening routers*. Beyond packet filtering, firewalls at network perimeters are conveniently located for several complementary services: NAT, VPN endpoints (Section 10.4), network monitoring, and logging (of full packets or headers only) to support audit, recovery and forensic services. Firewalls may be in a dedicated hardware unit or functionality in a multi-purpose device (e.g., server, router, IP switch, wireless access point). A *hybrid appliance* may provide multiple such functions plus intrusion detection and beyond-header inspection into payloads, i.e., *deep packet inspection* as is common in application-level filters (Sect. 10.2) to filter executables or malicious content. While hybrid appliances reduce the number of security boxes, *dedicated firewalls* also have advantages:

1. smaller attack surface: reduced functionality avoids often-exploited features, and simplifies "hardening" general-purpose devices;
2. specialist expertise: stand-alone firewalls are built, and in enterprise environments administered by, specialized experts (rather than generic administrators);
3. architectural features: devices custom-designed as firewalls may have hardware advantages over general-purpose hosts (cf. Section 10.2 dual homing, fast interfaces);
4. absence of regular-user accounts: these often prioritize usability over security, e.g., allowing authentication with user-chosen passwords, whereas mandatory two-factor authentication is more appropriate for hosts used exclusively by administrators;
5. physical isolation: dedicated inline devices provide defense-in-depth.

‡**PERSONAL AND DISTRIBUTED FIREWALLS.** With *host-based firewalls*, a built-in software-based firewall filters packets into and out of each individual host. All major operating systems support such *personal firewall* variants for end-user machines. An important use case is for hosts on untrusted networks (e.g., mobile devices in hotels, coffee shops, airports) beyond the protection of perimeter firewalls of enterprise networks or home-user cable modems. One default-deny approach involves user prompts ("Allow or deny?") on first instances of inbound and outbound access requests, with responses used to build allowlists and denylists to reduce further queries. Such personal firewalls allow user control of network access, at the cost of inconvenient prompts accompanied by details often insufficient for a user to make informed choices—raising difficult tradeoff issues. *Distributed firewall* variants for enterprise environments and servers involve centrally-defined policies distributed to individual hosts by an integrity-protected mechanism (e.g., digitally

signed). Overall, such *host-centric defenses* complement network-centric approaches, as hosts are not always topologically inside firewalls, nor can hosts rely on perfect protection from firewalls. Host-based protection alone leaves exposures while foregoing the ability of network-centric devices to detect and mitigate compromised hosts after the fact. Host-based firewalls are often combined with host-based intrusion detection systems.

‡**Exercise** (iptables). As part of Linux since 2001, the `iptables` user-space program (with corresponding kernel component) uses the built-in Netfilter framework to specify a firewall policy and implement packet filtering. Describe the main functionality of `iptables` in two pages or less. (Hint: [25, Ch.1].)

‡**Exercise** (Unencrypted tunnels). To allow HTTP (browsing), many firewalls allow outbound connections to port 80. However, standard networking *encapsulation* of one protocol by another—also called *tunneling*—can then be used as follows to bypass firewall policy. Use HTTP to carry, as payload, a second protocol that would be disallowed by firewall policy if used alone (as the outer protocol). Discuss why this obvious "hole" is not closed (hint: [8, p. 234–236] or [5, p. 36]). Note: encapsulation arises again in Section 10.4, and in discussion of IPsec in which case the payload may be encrypted.

10.2 Proxy firewalls and firewall architectures

PROXY FIREWALLS (INTRODUCTION). Packet filters primarily examine network and transport headers (e.g., IP, TCP). They are often combined with two other categories of firewalls discussed here. Each is a security intermediary between an internal client and an external service. Both may be viewed as proxies, in the sense of acting as beneficial middle-person agents that mediate end-party communications. However, they have different primary roles. *Circuit-level proxy* firewalls generically relay connections through a single proxy point; the primary mediation is to allow or deny the connection, and then relay bytes. *Application-level filters* carry out application-specific processing through multiple specialized processors for a pre-determined set of authorized applications. We discuss circuit-level proxies first, after a brief mention of important properties.

PROXY FIREWALL REQUIREMENTS. Two properties are critical in firewall proxies:

1. *transparency*. Ideally, the user experience is unchanged (as opposed to expecting compliance with special procedures).
2. *performance*. Performance degradation must be limited. Store-and-forward applications (e.g., email, file transfer) may be more tolerant than real-time applications.

Common means to achieve transparency from the client software experience are: a) *proxy-aware clients*—customizing selected client application software to redirect connections to a firewall proxy; and b) *proxy-aware gateways*—use of a primary firewall that redirects tasks to appropriate specialized processors based on application protocol fields.

CIRCUIT-LEVEL PROXIES (MOTIVATION). *Circuit-level proxy* firewalls arose in the context of early Internet gateways ("world wide web" proxies, Chapter 9). Enterprises desired to protect local hosts from malicious inbound connections, but also to allow internal users convenient web access (outbound connections). Toward this end, suppose

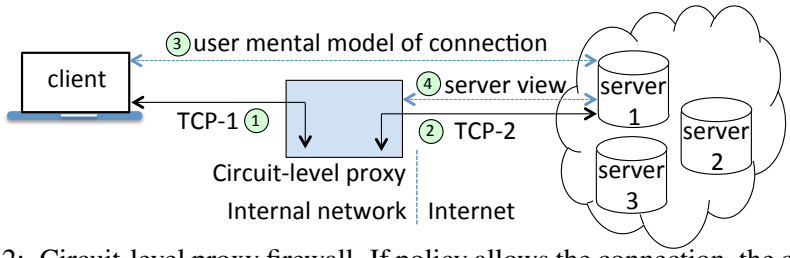

Figure 10.2: Circuit-level proxy firewall. If policy allows the connection, the circuit-level proxy establishes a virtual circuit that fulfills the user view of the connection. Technically, it receives packets on one TCP connection (TCP-1), with packet reassembly as usual, and retransmits using a second TCP connection (TCP-2) to the target server (server 1). From the server viewpoint, the connection is to the proxy, not the end-user client.

all within-enterprise hosts are on an isolated network without Internet connectivity, and considered "secure" for that reason; call them *internal hosts*. Consider hosts with Internet connectivity as "insecure"; call them *external hosts*. Enterprise security staff equated "security" with the lack of network connectivity between internal and external hosts; communication between the two was possible by manual transfer via portable storage media, but inconvenient. Thus users were unhappy. This led to the following manual solution.

MANUAL GATEWAY SOLUTION. A first-generation solution involved user accounts on a *gateway* machine. A host G (gateway firewall) is set up. It is connected to both the internal network and the Internet, but does not allow direct connections between the two.[5] Internal users are given accounts on G. Internet access is available by logging in as a user on G (but not directly from an end-user machine U). To copy files (transfer content) back to an end-user machine, now rather than using portable media, the physical transfer is replaced by a (manually initiated) network transfer. Users log in to their account on G, retrieve Internet content, have it stored on G; log out; then from their regular host U, establish a U–G connection allowing file transfer from G to U. Similarly, to transfer a file f from internal host U to external host X, a user (logged into U) copies f from U to G, and then (after logging into G) copies from G to X. Enterprise security staff remain happy, as there are still no direct connections from U to X. Users are less unhappy—they can do this all electronically from their desk, but it is time-consuming.

FIREWALL PROXY WITH PROXY-AWARE CLIENTS. An improved solution from the early 1990s remains the dominant variant of *circuit-level proxy* firewall (Fig. 10.2). It involves a client-side library, a proxy server (daemon) `sockd`, and a client–daemon network protocol called `SOCKS`. Collectively, they allow an internal user U to connect to a firewall-resident proxy `sockd` that selectively provides access to Internet content on external hosts X, with proxied path: U-to-`sockd`, `sockd`-to-X. The proxy is *transparent* in that U's experience is the same as an application-level connection directly to X, and no changes are required to external services (hosts X see `sockd` as the connection originator). This magic is possible by making pre-selected client applications `SOCKS`-aware,

[5] As noted later in this section, this is possible using a *dual-homed host*.

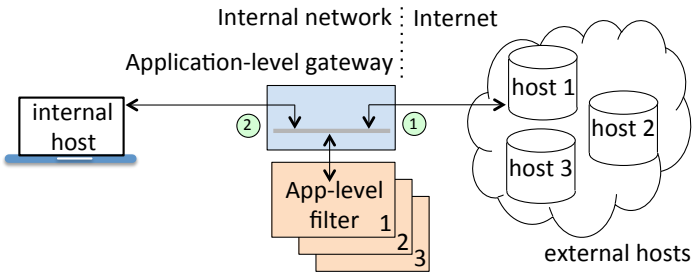

Figure 10.3: Application-level gateway filters. The application-level gateway selects the appropriate filter for application-specific filtering of data packets.

meaning: the client application software is modified such that its library routines that involve network sockets are replaced by corresponding routines from a `SOCKS` library, such that client outbound connections are redirected to `sockd` on *G*. Performance costs are low—on approving a connection and establishing a separate TCP connection with specified external host *X*, the proxy's main task is copying bytes from received packets of one completed proxy leg (TCP connection) to the other. Packets from the first TCP connection are reassembled before relaying.[6] Connection details (TCP socket details, number of bytes sent, username if provided) may be logged using `syslog` but unlike application-level filters (below), no further *content inspection* or filtering occurs.

CIRCUIT-LEVEL PROXIES: SUMMARY. The above mechanism delivers on the enterprise goal of safely facilitating outbound connections. Internal users have transparent access to Internet services consistent with an enterprise policy, without exposing internal hosts to hostile connections from outside. The main cost is customization of pre-selected client applications, and their distribution to internal users. As noted, circuit-level proxies may be combined with packet filters to block inbound connections except, e.g., from pre-authorized external sockets, or connections to restricted sets of internal sockets. The circuit-level proxy itself may require proxy connections be authenticated (so as to relay only policy-approved connections), beyond packet filter constraints on pre-approved sockets and protocol details (vs. application content). The proxy gateway is mandatory for connections—endpoints are on networks not otherwise connected. Note that circuit-level proxies themselves (including `SOCKS`) do not provide inherent encryption services.[7] Beyond proxying TCP connections, `SOCKS` also supports forwarding of UDP packets.

APPLICATION-LEVEL FILTERS. As noted earlier, *application-level gateways* filter traffic using specialized programs for a pre-determined set of applications (Figure 10.3). These specialized programs may be considered "proxy processors" (but execute different tasks than circuit-level proxies). Packets corresponding to a targeted application protocol are directed to the appropriate customized filter, each of which leverages detailed understanding of a specific protocol to examine application-specific data (cf. Figure 10.14). This may result in not only blocking packets entirely, but altering payloads—e.g., *con-*

[6] This also mitigates some (beyond our scope) exploits involving intentional packet fragmentation.

[7] `SOCKS` is, however, often combined with an encrypted tunnel, e.g., provided by SSH (Section 10.3).

tent inspection for an HTTP gateway/proxy may involve deleting malicious JavaScript or rewriting URLs. Such "layer 7 firewalls" that alter payloads do intrusion prevention (Chapter 11).

APPLICATIONS TARGETED. The requirement for detailed knowledge of specific applications limits the number of protocols for which application-specific filters are built, and raises issues for proprietary application protocols whose details are not widely known. Targeted applications include those most widely used, and those causing the biggest security problems. Among the first were specialized filters for remote login, file transfer, and web protocols (TELNET, FTP and HTTP). Email is in its own category as a primary application to filter, and (independent of firewall technologies in use) is commonly processed by multiple mail-specific gateways on the path from originator to recipient, including to filter spam and to remove malicious embedded content and attachments. Mail filters may remove Microsoft Word macros (cf. macro viruses, Chapter 7), or executables. Regarding performance, examining application-level content implies longer processing times, partially mitigated by architectures with dedicated processors per application type.

BASTION HOSTS AND DUAL-HOMED HOSTS. Firewalls that serve as gateways between internal subnetworks are called *internal firewalls*. These (and DMZs below) support the principle of DEFENSE-IN-DEPTH (P13), allowing management of sensitive subnetworks, e.g., to isolate (contain) test/laboratory networks. Borrowing terminology from medieval castles–where a *bastion* is a fortified cornerpoint or angled wall projection such that defensive firepower can be positioned in multiple directions—a *bastion host*, in a multi-component firewall, is a defensive host exposed to a hostile network. While designed to protect the internal network, it is itself exposed (e.g., behind only a screening router), and thus should be *hardened* (locked down) by disabling all interfaces, access points, APIs and services not essential to protecting the internal network or facilitating controlled Internet access for internal hosts. A *dual-homed host* is a computer with two distinct network interfaces, and correspondingly two network addresses; multi-homed hosts have one IP address per interface. If routing functionality between the two interfaces is disabled, a dual-homed host is suitable for a circuit-level proxy—physically ensuring the absence of direct connections between an external and internal network. A dual-homed host may serve as a single-host (one-component) firewall to an external network, but more commonly is part of a multi-component firewall architecture.

ENTERPRISE FIREWALL ARCHITECTURES. As a minimally functional enterprise firewall, a single screening router (router with packet filtering) offers basic protection but limited configurability. A slightly more functional firewall has a screening router and a bastion host. The screening router is on the Internet-facing side of the bastion host, which protects the internal network and receives incoming connections (e.g., email and DNS queries about enterprise addresses). A more comprehensive architecture, providing an outer layer in a DEFENSE-IN-DEPTH (P13) strategy, uses a *perimeter network* called a *network DMZ* (demilitarized zone)—a subnetwork between an external network (hostile) and the internal network to be protected. To follow the principle of LEAST-PRIVILEGE (P6), the types of traffic (protocols) allowed within the DMZ should also be minimized. One such architectural design (Figure 10.4) consists of a bastion host between a first

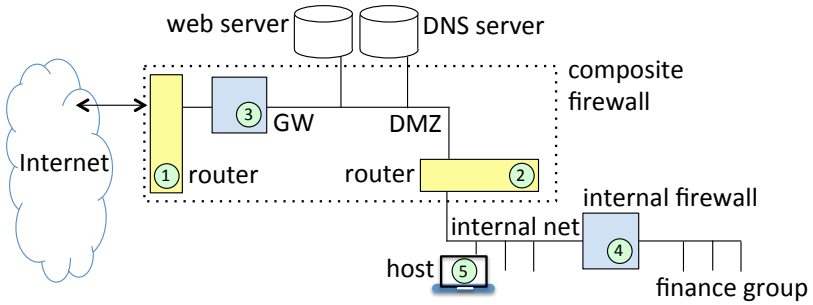

Figure 10.4: Firewall architecture including DMZ. The gateway firewall (3) is a bastion host. An internal host (5) connects to Internet services by proxying through GW, and might be allowed to make outgoing connections (only) through the exterior router (1), bypassing GW, for a reduced set of packet-filtered protocols, depending on policy.

screening router (exterior router) facing the external network, and a second screening router (interior router) fronting the internal network. The bastion host in the DMZ is the contact point for inbound connections, and the gateway proxy for outbound connections.

‡**Exercise** (Firewall architecture details). Give four relatively simple firewall architectures, from a single screening router to that of Figure 10.4, and the advantages of each over the earlier (hint: [41, Ch. 6]).

10.3 SSH: Secure Shell

The Internet Protocol (IP), the Internet's foundational datagram networking protocol, lacks built-in security services. Yet higher-level protocols sometimes implicitly (at times, falsely) assume that protection comes from lower-level protocols. One possible, but inefficient, solution would be for each application to independently implement its own security services. A more efficient one—albeit requiring coordination and advance planning—is to build generic security services into lower levels of the communications protocol stack (Fig. 10.10 on page 300), for higher levels (especially applications) to rely on. This is the approach followed by SSH, introduced in 1995 and now widely used worldwide. Though SSH stands for *Secure Shell*, SSH does not itself provide a shell, but rather provides an encrypted tunnel to get to a shell (and other programs).

SSH OVERVIEW. Prior to SSH, means for remote login (`rlogin`, `telnet`), related Unix *remote-access commands* (`rsh`, `rcp`, `rexec`), and file transfer (`ftp`) sent cleartext data over the network, and also passwords (when required). In some cases, password authentication for remote access was not required (page 297, "trusted" login hosts). SSH, and utilities built using it, were specifically designed as secure alternatives (with confidentiality, integrity, authentication), as summarized in Table 10.2. Using TCP for reliable packet transport, SSH provides a *security tunnel* by its own transport layer protocol, protecting both login passwords sent to remote services, and other data to be transported by TCP. The general model of SSH is that any program available on a remote host can be run through the security tunnel. SSH is now widely used to secure both custom-built utilities

TARGET	FULL NAME	FUNCTIONALITY
`rsh` (`ssh`)	remote shell	Send shell commands for execution on remote host by a daemon (`rshd`).
`rlogin` (`ssh`)	remote login	Log in to remote Unix server over TCP network, then communicate as if physically local.
`telnet` (`ssh`)	teletype network	Acquire interactive virtual terminal connection over TCP, e.g., to a command line interface (Unix, Windows).
`ftp` (`sftp`)	file transfer prot (secure ftp)	Transfer files, using separate connections for control and data. Can also be replaced by `ftps` (FTP over TLS).
`rcp` (`scp`)	remote copy (secure copy)	Copy files, directories between local, remote systems, with command line syntax similar to Unix command `cp`

Table 10.2: Widely used remote access command programs for network protocols prior to SSH. Secure replacements (parenthesis, column 1) are available in the OpenSSH open-source suite and other packages. The SSH precursors are also called *Berkeley r-utilities*.

(e.g., `scp`, `sftp`), and general access by a local host to remote services.

REMOTE SHELL VIA SSH. Figure 10.5 depicts use of SSH for a *remote shell*. The user (local host) ends up with a terminal interface and command prompt for an interactive shell running on the remote machine. The shell may be thought of as using standard input/output streams (`stdin`, `stdout`, `stderr`) to communicate with the SSH daemon (`sshd`), which relays traffic to SSH client software (`ssh`). Thus `ssh` and `sshd` work as partners at client and server sides (local and remote ends). By this design, remote applications need not be customized to work with SSH (i.e., need not be "SSH-aware").

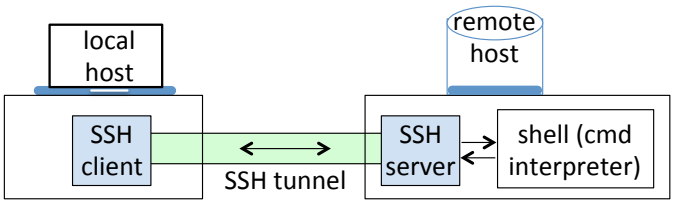

Figure 10.5: Remote shell through an SSH tunnel, providing authenticated encryption.

THREE PROTOCOLS COMPOSING SSH. SSH is implemented in three parts:

1. Its *transport layer protocol* provides SSH server authentication, encryption and integrity protection. It includes negotiation of cryptographic parameters and keys.
2. Its *user authentication protocol* handles SSH client-to-server authentication and runs over the transport layer protocol, relying on it for confidentiality and integrity.
3. Its *connection protocol* enables use of a single SSH connection (encrypted tunnel) for multiple purposes. Each use instance is assigned a number denoting a "logical channel"; data for all channels is sent over the single SSH connection and the channels are said to be *multiplexed*. The connection protocol, running over the transport layer protocol and after the authentication protocol, employs the encrypted tunnel. Channels may support *interactive sessions* for remote execution of programs (shells, applications, remote execution of system commands) and *connection forwarding* (of ports

and X11 sessions, below). After a channel is defined (set up), a single remote program can be started and associated with it.

SSH CLIENT AUTHENTICATION. During session negotiation, the SSH server declares which client authentication methods its SSH clients may use. While additional options may be supported, common options include the following:

- client password (static, or one-time passwords);
- Kerberos ticket obtained from a Kerberos server (Chapter 4); and
- client public key (described next).[8]

A typical *SSH client authentication* process for the case of client public keys is as follows:

i) The server receives from the client the client public key, as well as the client's signature (using the matching private key) over pre-specified data including a session ID from the transport layer.

ii) The server checks that two conditions hold—the client public key is on a list of server-recognized (pre-registered) keys; and the received signature is valid relative to this public key and the data including session ID.

SSH SERVER AUTHENTICATION: ESTABLISHING TRUST IN HOST KEY. Server authentication involves a server public key (*SSH host key*), and standard protocols for using a recognized public key for authenticated key establishment.[9] This requires that the client recognize (trust) the host key. In non-business uses generally lacking supporting infrastructure, a relatively weak common approach is *trust on first use* (blind TOFU, Chapter 8); this provides protection against passive attackers. To also preclude active middle-person attacks (where an attacker substitutes a fraudulent host key), the end-user can cross-check the *fingerprint* (Section 8.2) of the host key, e.g., manually check that a locally computed hash of the offered host key matches a hash thereof obtained over an independent channel. (Enforcement of such a check is problematic in practice; many users skip this step even if asked to do it.) In business uses, where a requirement for higher security justifies added costs, an enterprise may make such cross-checks corporate policy and take steps aiming to increase user compliance, but the preferred alternative is an infrastructure-supported (automated) method to trust SSH server keys (Model 2 below).

TRUST MODELS FOR SSH HOST KEYS. The above motivates two models:

- Model 1 (client database of SSH server keys). On first connection to an SSH server, a client is offered a server host key. If the client manually accepts it as trusted (with or without a cross-check), the key, paired with a server ID (e.g., URL or domain) is stored in a local client database for future use as a "trusted" SSH server key.
- Model 2 (CA-certified SSH server keys). Here one or more CA verification public keys are configured into the SSH client (perhaps via a local file), and used to verify offered SSH host keys. This resembles TLS use of the CA/browser trust model (Chapter 8), but there, client (browser) trust is pre-configured by browser vendors, and generally

[8] Conformant software must support the client public-key option, but all clients need not have public keys.
[9] TLS set-up (Chapter 9) similarly uses a recognized server public key to establish a fresh session key.

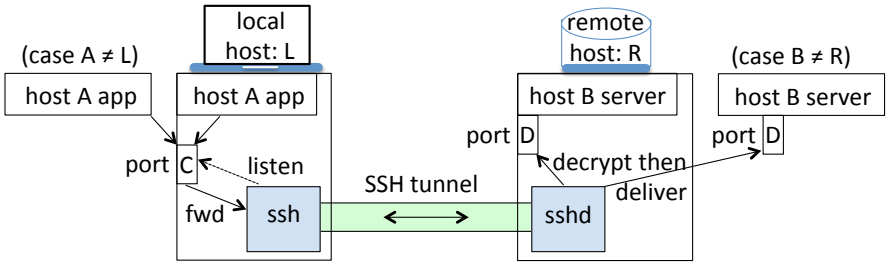

Figure 10.6: SSH local port forwarding. An application on host *A* historically connects over the Internet to port *D* on a distinct host *B*. To secure this, an SSH tunnel is set up between `ssh` on host *L* and `sshd` on host *R*, with `ssh` configured to listen on a local port *C*, which the app is then directed to send data to; such data ends up being received by `sshd` and forwarded as desired. A mnemonic mapping of the command-line syntax to this diagram is: `ssh -L portC:hostB:portD hostR`. By default, `ssh` assumes that the application using the SSH tunnel is on the same host as the `ssh` client, i.e., $A = L$.

designed to facilitate access to a maximum number of sites by a maximum number of users. In contrast for SSH, an enterprise or other host of an SSH server may intend access for only a restricted set of authorized users (e.g., employees and partners), and thus any CA infrastructure used may be far less global (and easier to manage).

‡**SSH USER AUTHENTICATION (METHOD CONFUSION).** When the client public-key method is used, the matching private key is often stored in software at the client, protected under a passphrase or password-derived key. The user would then typically be prompted to enter a password to allow private-key access. Depending on the user interface, this may cause confusion on whether user authentication employs password or public key.

SSH PORT FORWARDING. SSH uses *port forwarding* of TCP ports to redirect data from unsecured (e.g., legacy) applications through a pre-established SSH connection (SSH tunnel). The application would otherwise send cleartext over a TCP/IP connection. To understand port forwarding (Fig. 10.6), consider the command syntax:

```
ssh -L listen_port:host:hostport sshd_host
```

This invokes the local client (`ssh`) to set up an SSH connection to a daemon (`sshd`) on `sshd_host`, establishing the SSH tunnel. The "`-L`" option and its argument results in `ssh` listening for data sent to `listen_port` on the local host, and sending it through the SSH tunnel and then on to its final application destination, `host:hostport`. The application wishing to use the tunnel must then be configured to send data (connect) to `listen_port`. That data, now redirected through the `ssh`-`sshd` tunnel from the local host to `sshd_host`, is decrypted by `sshd`, and relayed to `host:hostport` (which in the simplest case is also on `sshd_host`; otherwise, now-plaintext data is no longer SSH-protected over its final leg, e.g., on the local network), where the application data is consumed. This is (L) *local port forwarding*, or *tunneling a port*. The analogous process to request forwarding of ports on a remote SSH host is called (R) *remote port forwarding*.

‡**Exercise** (Remote port forwarding). Draw a diagram like Fig. 10.6 for remote port forwarding, and explain the corresponding command-line command (hint: [27, p. 224]).

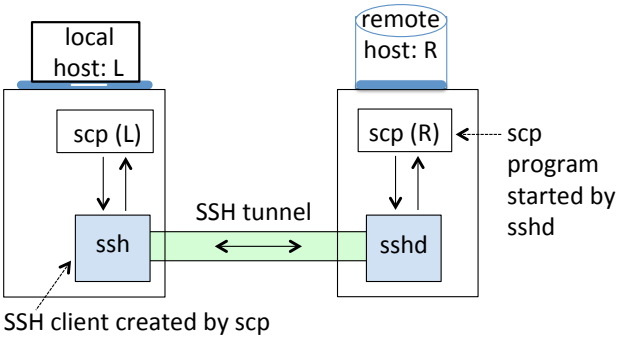

Figure 10.7: SCP (secure copy). `ssh` is client software; `sshd` is the server daemon. Both programs are supported on most operating systems, and are what users commonly associate with "SSH" (although SSH is actually a set of network protocols). SCP was custom-designed to replace `rcp`; other applications use SSH by explicit port forwarding.

How SCP works. Figure 10.7 illustrates the design of SCP. For a user on host L to transfer *file1.txt* to host R using SCP, the syntax is:

```
scp file1.txt R
```

This results in the SCP software client `scp` (L) forking an SSH child process on L; issuing to it an SSH command starting a remote copy of SCP on R; and with that command, sending R also the embedded command: `scp -t file1.txt`.[10] On R, a daemon `sshd` is listening and a copy of `scp` (dormant) is available. The coordinating pair `ssh`-`sshd` set up an SSH tunnel, and the local and remote SCP agents use the tunnel to transfer the file.

‡**SSH X11 forwarding.** SSH likewise supports forwarding of X11 connections. This refers to a windowing system for bitmap displays, the X Window System (version 11), specifically designed for use over network connections to facilitate remote graphical user interfaces (GUIs). An X server, running on a user's local machine with graphical display resources (and keyboard and mouse), is used for local display and input-output for an X (client) program running on a remote (network) computer. X allows a program to run on a remote machine with GUI forwarded to a machine physically in front of a user.

‡**Exercise** (SSH host-based client authentication). A further SSH client authentication method specified in RFC 4252 is *host-based* (client) authentication. It involves a client-side signature, in this case using a private key for the client host machine as a whole (rather than corresponding to a specific user thereon); the server must, in advance, have information allowing it to recognize the client public key as authorized. This method is sometimes combined with requiring that the client is also allowed to log in to the server according to a *.rhosts* file on the server (per "trusted" login hosts, below). Discuss advantages and disadvantages of this method. (Hint: [37]; for more insight, [41, p.502].)

‡**Exercise** (SSH and secure file transfer). In a table, briefly summarize and compare the security-related properties and functionality of the following:

[10] The `-t` flag is typically omitted from user documentation, as it is not meant for use by end-users. The flag tells the receiving `scp` program that it is a remote `scp` instance that will be receiving a file to be stored using the specified filename.

a) historical Simple File Transfer Protocol (first SFTP, per RFC 913)
b) `rcp` of old Unix systems
c) `scp` (SSH replacement of `rcp`)
d) `ftp` and its TLS-based extension `ftps` (RFC 4217)
e) `sftp` (i.e., SSH FTP, the second SFTP, beyond that of part a)

‡**Example** *(PuTTY: SSH client tools).* PuTTY is a popular open-source package available for most operating systems. The core functionality is secure remote sessions, via SSH replacements of the standard network protocols (`rsh`, `telnet`, `rlogin`), typically also including `scp` and `sftp`—thus all the secure replacements listed in Table 10.2.

‡**"TRUSTED" LOGIN HOSTS: MECHANISM.** Historical tools (with Unix origins) sent cleartext passwords for remote login (`rlogin`) and remote execution of commands (`rsh`). For convenience, passwords could be omitted for connection requests from hosts (generally in the same local network) designated as "trusted" in a root-owned or per-user special file on the machine being connected to. Given a list of hosts in */etc/hosts.equiv*, requests asserted to be from one of these were given access with the authority of the local userid (if it exists) matching the userid asserted. For <remote_host, remote_user> pairs included in the home directory file *.rhosts* of an individual user, requests from such a pair were granted access under the local userid of that individual. (Host-userid pairs could also be specified in */etc/hosts.equiv*, but this was uncommon.)

‡**"TRUSTED" LOGIN HOSTS: DANGERS.** The above trusted-hosts mechanism is now strongly discouraged and often disabled, but remains useful to know both as an example of a risky practice, and to understand the context in which SSH was created. The mechanism gives, to an attacker having gained access to one account, password-free access to other machines that, by this mechanism, trust the first account (or its host). Such other machines themselves may be trusted by a further set of machines, and so on. Such (pre-granted, unauthenticated) *transitive trust* breaks principle P5 (ISOLATED-COMPARTMENTS), and compounds the failure to follow P6 (LEAST-PRIVILEGE). As a concrete example, this trust mechanism was exploited by the *Morris worm* (Chapter 7).

‡**PORT 22 FOR SSH.** SSH inventor Tatu Ylönen requested port 22 be dedicated for SSH, and the request was granted. He relates:[11] *I wrote the initial version of SSH (Secure Shell) in Spring 1995. It was a time when telnet and FTP were widely used. Anyway, I designed SSH to replace both telnet (port 23) and ftp (port 21). Port 22 was free. It was conveniently between the ports for telnet and ftp.*

10.4 VPNs and encrypted tunnels (general concepts)

MOTIVATION: PLAINTEXT PACKETS. Normal TCP/IP packets are plaintext. The full packet content (header plus payload) is visible to every party with access to the packet stream (open wireless links make eavesdropping trivial), and alterable by any inline party (all intervening routers, switches, gateways, and service provider equipment). For protection, one idea is to encrypt entire packets at origin devices before network transmission.

[11] Source: `https://www.ssh.com/ssh/port`

However, that breaks existing networking protocols, which rely on plaintext header fields to allow packet processing, forwarding and delivery. Thus if packet header fields are encrypted, the encrypted data must be repackaged as a payload, preceded by a new header that can in turn be removed by networking software at the destination. Alternatively, payload data alone can be protected (e.g., by authenticated encryption), in which case existing networking protocols are not disturbed. This leads to strategies including tunneling.

TUNNELING. In the networking context, *tunneling* refers to one data stream's journey (the inner) being facilitated by another; the imagery is of a tunnel. Contrary to standard network stack protocol design, where protocols lower in the stack (Fig. 10.10, p.300) carry payloads of higher-level protocols (Fig. 10.14), tunneling may also involve one protocol carrying a same-level protocol. The technical means is (as in standard protocol design) *encapsulation* of one protocol by another—a first protocol (header plus payload) is the payload of a second, the second prefixing a new (outer) header. Viewing the first protocol as having two parts, a letter body surrounded by an envelope (which serves to provide a final address), encapsulation puts a second envelope (with interim destination) around the first. Not all tunnels provide security, but security tunnels allow secure transit via public/untrusted channels (Fig. 10.8). Two widely used technologies often viewed as security tunnels are the relatively lightweight SSH (Sect. 10.3), and heavier-weight IPsec (Sect. 10.5). The idea is that once a tunnel is set up, applications (and their users) transparently enjoy its security benefits without requiring or experiencing changes to those existing applications; security-related details disappear by the time the application data is consumed. Encrypted tunnels set up this way are used to secure data (including from legacy protocols) that transits untrusted networks, and for VPNs as discussed next.

VIRTUAL PRIVATE NETWORKS. A (physical) *private network* is a network intended for access only by trusted users, with security (e.g., confidentiality, integrity) relying on a network architecture providing physical isolation. Examples are local area networks internal to an enterprise, and home networks meant for private use. A *virtual private network* (VPN), often used by enterprise organizations, is a private network typically uniting physically distant users or subnetworks, secured not by physical isolation but use of encrypted tunnels and special-purpose protocols (e.g., authentication-based access control), software, and hardware including firewalls or gateways. *Virtual* here suggests use of shared public links (secured by cryptography), whereas historically private networks involved costly exclusive-access leasing, from telecommunications companies, of dedicated

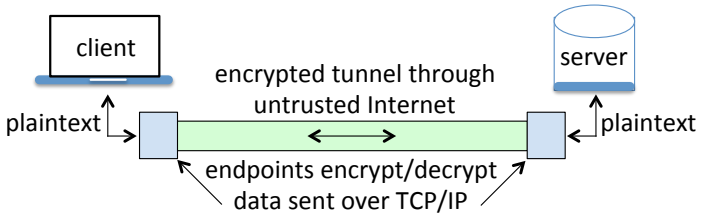

Figure 10.8: Encrypted tunnel (concept). To avoid breaking pre-existing protocols, the tunneling protocol must preserve packet header data used for routing and middlebox (i.e., non-endpoint) processing.

VPN design	VPN architecture	Notes and use cases
transport mode	host-to-host VPN	provides end-to-end security (VPN endpoints are final destination)
tunnel mode	network-to-network	network gateways add/remove VPN security (no VPN protection internal to gateway)
	host-to-network	for remote host access to enterprise (in-host gateway adds/removes VPN security)

Table 10.3: VPN designs and architectures. See Fig. 10.9 for illustrations.

network links (communications cables) physically connecting remote networks.

VPN USE CASES. Two primary use cases are met by the designs in Table 10.3:

1. *site-to-site VPNs*. The idea is to bridge private networks across a public channel.
2. *remote access VPNs*. The idea is to allow authorized clients remote access (e.g., from home) to a private network, with access experience as if physically local.

LIMITATIONS OF ENCRYPTED TUNNELS. Encrypted payloads prevent effective network-based monitoring, and content-based filtering at gateways and firewalls. With plaintext, detection (and altering or dropping) of packets containing malicious data patterns is possible; encrypted such patterns cannot be detected. In the case of an *insider* attack (a disaffected employee, or an earlier compromise giving an attacker access to an internal-network machine), data exfiltrated through an encrypted tunnel is no longer easily detected by standard network monitoring as in the case of plaintext. Also, if a firewall allows transit of, e.g., SSH packets, a protocol that a firewall policy blocks directly may still be sent as data through an SSH-provided tunnel (evading firewall policy). Use of encrypted tunnels, including IPsec, thus increases the importance of host-based defenses.

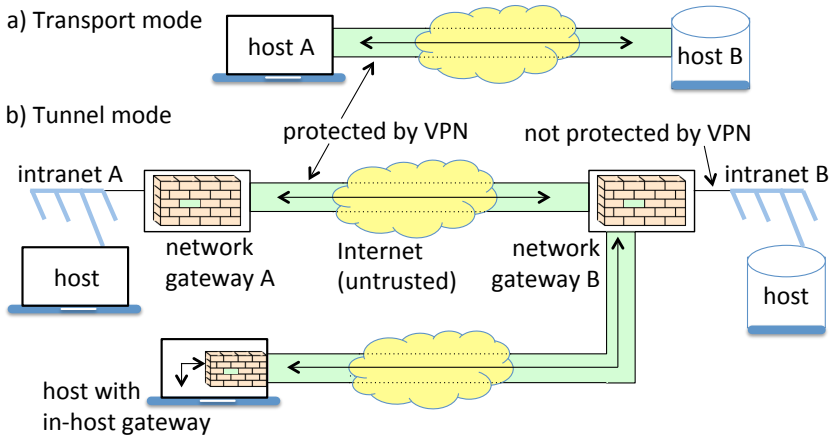

Figure 10.9: VPN designs. (a) Transport mode is host to host (single hosts), still delivering a payload via an encrypted tunnel in the sense of Fig. 10.8. (b) Tunnel mode involves network gateways. In the in-host gateway case, one end has a within-host final hop. Intranet A, on the enterprise side of a gateway, is an internal enterprise network. Intranet B may be a second enterprise network, or a remote employee's home network.

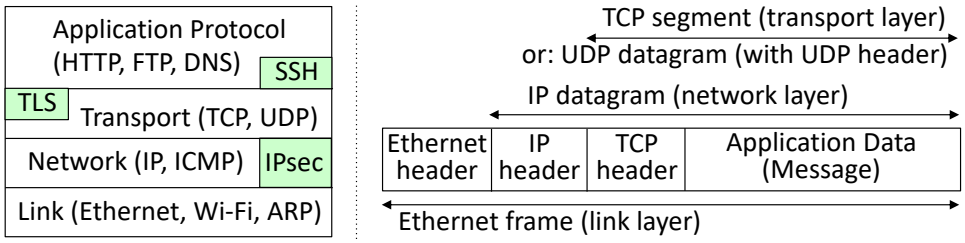

Figure 10.10: Network protocol stack (TCP/IP model) and encapsulation. In the seven-layer OSI model, between Application (7) and Transport are Presentation and Session layers, and Link is Data Link, above Physical (1). Wi-Fi denotes IEEE 802.11 wireless.

NETWORKING CONTEXT. An understanding of how network protocols are secured is aided by reviewing the basic framework of network protocols. Figure 10.10 reviews the conventional network communications stack, complementing Section 10.6. For perspective, the figure shows where SSH, IPsec, and earlier-discussed TLS protocols fit in.

10.5 ‡IPsec: IP security suite (details)

The Internet Protocol (IP) provides no security services—no encryption, no authentication even of IP header addresses. TLS provides security services to application-layer protocols when the latter are invoked such that they use TLS; likewise SSH client software can be used to set up SSH tunnels to secure other application protocols. In contrast, the motivation behind the *IPsec* (IP Security suite) protocols is to provide network-layer security services (Fig. 10.10) that are automatically inherited by all transport and application layer protocols. IPsec enables VPNs through a broad and flexible suite of security services delivered by three protocols: IKE for key management, AH for authentication only, and ESP (which includes encryption as well as authentication options). We now describe each.

IKE: INTERNET KEY EXCHANGE. While IPsec supports manual key management by system administrators, its IKE component automates key establishment using Diffie-Hellman. In IPsec, like many protocols, data transfer is preceded by a parameter negotiation stage setting up protocol details between endpoints. The resulting shared state (agreed algorithms, sequence numbers, cryptographic keys) defines a *security association* (SA). A separate SA is used for each communication direction. Each SA is indexed by an SPI (security parameters index); IPsec headers include an SPI field, as discussed next.

AH: AUTHENTICATION HEADER. Figure 10.11 lays out the AH fields in an IP packet. The AH component provides a MAC for data origin authentication (attribution) of the entire IPsec payload plus those IP header fields unchanged by routers en route (immutable); fields designated as *mutable* (e.g., TTL) are zeroed for the purpose of MAC computation. AH also optionally provides replay protection, using the AH sequence number field plus other details from stored security association data identified by the SPI field.

ESP: ENCAPSULATING SECURITY PAYLOAD. The ESP component allows encryption of the IPsec payload, plus services similar to AH (i.e., a MAC for attribution,

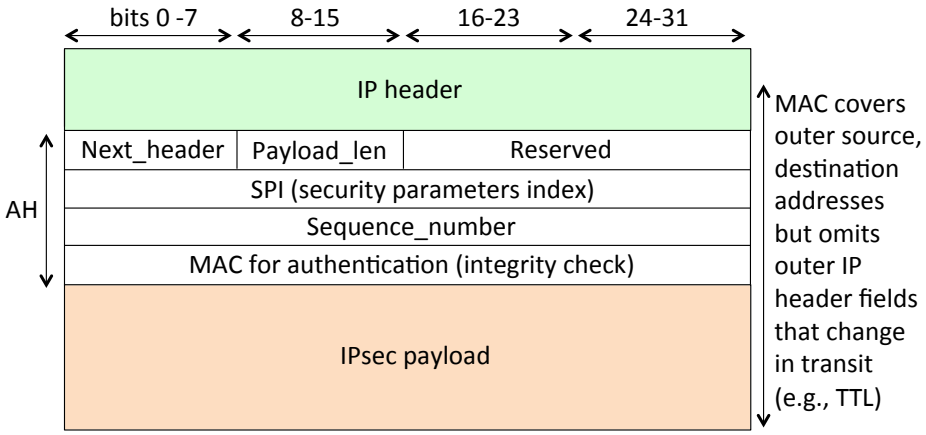

Figure 10.11: IPsec Authentication Header (AH) field view, for both transport and tunnel modes. `Next_header` identifies the protocol of the AH payload (e.g., TCP=6). `Payload_len` is used to calculate the length of the AH header. `SPI` identifies the Security Association. `Sequence_number` allows replay protection (if enabled).

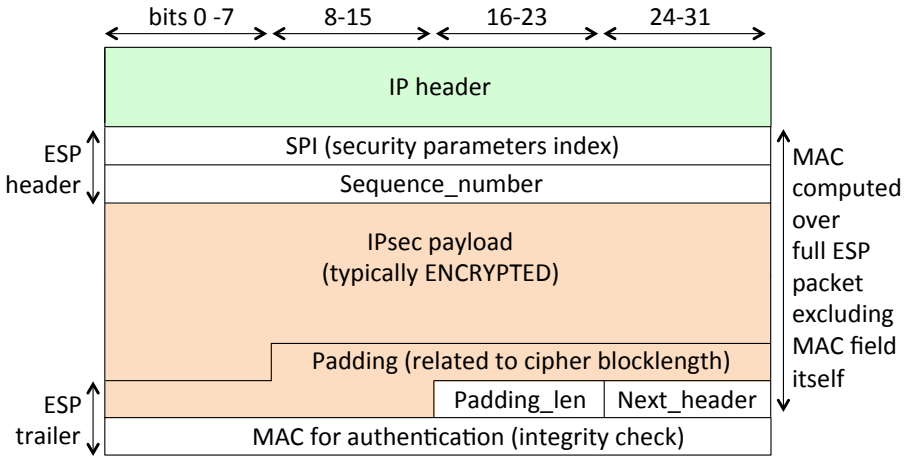

Figure 10.12: IPsec Encapsulating Security Payload (ESP) field view, for both transport and tunnel modes. `SPI` identifies the Security Association. `Sequence_number` allows replay protection (if enabled). `Next_header` (which may include a crypto IV or Initialization Vector) indicates the type of data in the ENCRYPTED field. A payload length field is not needed, as the ESP header is fixed at two 32-bit words, and the length of the IPsec payload (which is the same as that of the original payload) is specified in the IP header.

and optional replay protection). However in the ESP case, the MAC does not cover any IP header fields. Figure 10.12 shows how ESP fields are laid out within an IP packet.

IPSEC: TRANSPORT MODE. AH and ESP can each operate in two modes (Table 10.3, page 299). IPsec *transport mode* is used to provide an end-to-end VPN from one host to another host. As shown in Fig. 10.13b, for *transport mode* the IPsec header is inserted between the original IP header and the original IP payload. Here the original

IP header is *transported* (but not tunneled), e.g., when used with ESP, the original IP payload (but not the original IP header) is encrypted as the "IPsec payload". Note that *transport mode* cannot be used if one endpoint is a network, as the resulting IPsec packet has only one IP header per Fig. 10.13b; thus there would be no IP address available for a second-stage delivery (after using the one destination IP address to reach the network gateway).

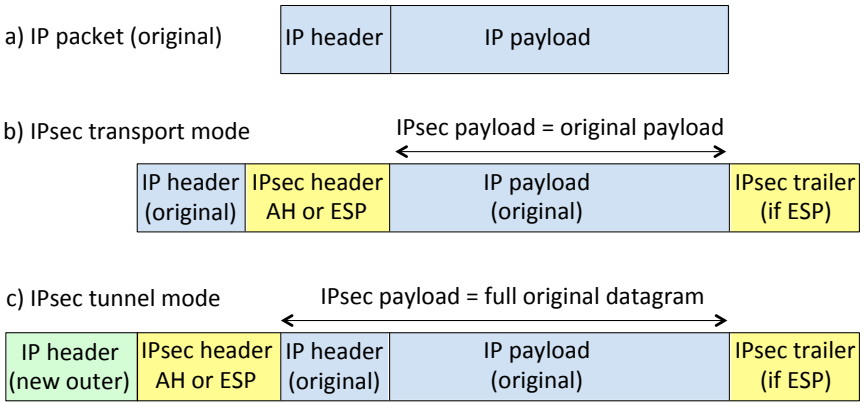

Figure 10.13: IPsec transport mode vs. tunnel mode (structural views).

IPSEC: TUNNEL MODE. IPsec *tunnel mode* has two VPN use cases (cf. Table 10.3, page 299): network-to-network VPNs, and host-to-network VPNs.[12] In the first case, the VPN terminates at security gateways to an enterprise network at each side. The gateways are the endpoints with respect to AH or ESP protection; packets are unprotected for the remainder of their journey within the enterprise network. Thus end-to-end security is not provided. The delivering gateway forwards the inner packet to its end destination (host), using the remaining IP header (inner packet) after the outer IP header and IPsec header/trailer are consumed (removed) by the gateway. In the second case, VPN functionality built into the remote host software functions as an "in-host network gateway". Figure 10.13c shows the IPsec packet structure for *tunnel mode*: the entire original IP datagram (including the IP header) becomes the IPsec payload, preceded by an IPsec header, preceded by a new (outer) IP header. Thus there is *encapsulation* of the (entire) original IP datagram, i.e., *tunneling*. In particular, this is an IP-in-IP tunnel.

‡**Exercise** (IPsec anti-replay). a) Explain how sequence numbers in AH and ESP headers are used in a *sliding receive window* method for IPsec's anti-replay service (hint: [17], or [27, pages 328–330]). b) Explain why this network-layer anti-replay mechanism is more complicated than use of (implicit) sequence numbers for anti-replay in SSH, where datagrams are carried with TCP delivery guarantees (hint: [27] again).

‡**IPSEC CHALLENGES AND DEPLOYMENT.** IPsec's configuration options offer great flexibility. In turn, IPsec is described as "heavyweight" with corresponding disadvantages: (a) Its large code base, options and complexity imply that running an IPsec VPN

[12] AH and ESP can each operate in tunnel mode or transport; a packet-level view is shown in Fig. 10.13.

typically requires a dedicated expert. (b) Deployment requires substitution of network-level stack functionality. Deployment approaches for achieving the latter are:

1. *OS kernel build-in*—incorporating IPsec into the host's core TCP/IP network stack.
2. *bump in the stack*—inserting a shim layer between existing network and link layers.
3. *bump in the wire*—deploying IPsec through introduction of inline hardware.

This is more complicated than creating a security tunnel by an end-user calling a software application (e.g., as in SSH or TLS); this explains why IPsec is called *heavyweight*.

‡**IPsec policies.** Once an IPsec VPN is in place, not every datagram addressed to a configured VPN peer is necessarily sent through the relevant VPN. A datagram must first meet a policy rule of the relevant *IPsec policy*. This allows policy-based packet filtering. Rules may depend on fields in IP and transport-layer headers (e.g., destination address and port, source address and port, protocol), or other conditions as specified within IKE (e.g., user identity). A host maintains such policies in an IPsec *security policy database*.

‡**ESP configurations.** Among ESP configurations are an authenticate-only ESP mode (using *null encryption*), an encrypt-only mode (discouraged as insecure), and ESP protection of a datagram followed by AH. As ESP provides all AH functionality except authentication of some outer IP header fields, many VPNs run ESP without AH.

‡**Exercise** (NAT incompatibilities; utility of AH). NAT (Section 10.1) modifies IP addresses in IP packet headers. a) Summarize incompatibilities this raises with AH integrity protection of headers, and issues NAT raises with ESP (hint: [22, 2]). b) Summarize arguments for and against the view that AH is unnecessary, as ESP can more efficiently protect against essentially all relevant threats (hint: [27, 17, 18]).

‡**Exercise** (WireGuard). WireGuard is a VPN alternative to IPsec and OpenVPN, with focus on simplicity, usability and speed. Summarize its major architectural design choices, and discuss pros and cons of its "crypto versioning" strategy (hint: [9], [10]).

10.6 ‡Background: networking and TCP/IP

IP and addresses. The Internet Protocol (IP) is the main protocol in the TCP/IP protocol suite for packet-switched networks. Figure 10.10 (page 300) illustrates the *network stack* framework used for network communications software. An *IP address* is a *logical* address identifying an (addressable) interface for data delivery to an IP host device, used in network routing; it identifies a host's current network location, rather than a physical host. IPv4 addresses are 32-bit numbers written as dot-separated 8-bit groups. IPv6 addresses are 128 bits. At the link layer, e.g., on a *local area network* (LAN), *physical* addresses identify hosts and are used for delivery of data *frames*. IP addresses are mapped to physical *MAC addresses* (*media access control* here, not *message authentication code*) using the Address Resolution Protocol (ARP, Section 11.5), which broadcasts messages on the link layer. ARP messages are local to a LAN (they do not cross routers).

Datagrams and packets. Packet-switched networks transfer data between hosts through "hops" between intermediate network devices (e.g., routers). Delivery involves a *datagram* composed of a header (to facilitate delivery), and a payload (the data intended

for the recipient). A 16-bit length field allows an IP datagram up to 65,535 bytes. Physical networks composing each hop deliver data in units called *packets* with size limit denoted by a *maximum transmission unit* (*MTU*), e.g., the maximum Ethernet payload is 1500 bytes. A datagram exceeding an MTU will be broken into *fragments* that fit within data *frames*; for reassembly, a *fragment offset* header field indicates the offset into the original datagram. Thus not all datagrams can be sent as single packets, but each packet is a datagram that can be independently delivered, and a packet may be a fragment of a larger datagram.

TCP AND UDP. IP datagrams are the network-layer means for transmitting TCP, UDP, and (below) ICMP data. TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) are distinct transport-layer protocols for transferring data between hosts. UDP is termed *connectionless* (as is IP): it provides unidirectional delivery of datagrams as distinct events, with no state maintained, no guarantee of delivery, and no removal of duplicated packets. In contrast, TCP is *connection-oriented*: it provides reliable bi-directional flows of bytes, with data delivered in order to the upper layers, and if delivery guarantees cannot be met, a connection is terminated with error messages. A TCP *segment* is the payload of an IP datagram, i.e., the content beyond the IP header (Fig. 10.14; compare to Fig. 10.10 on page 300). The TCP payload data in turn is application protocol data, e.g., for HTTP, FTP, SMTP (Simple Mail Transfer Protocol).

PORTS AND SOCKETS. *Ports* allow servers to host more than one service; the transport layer delivers data units to the appropriate application (service). A port number is 16 bits. A server offers a service by setting up a program (network daemon) to "listen" on a given port and process requests received. On some systems, binding services to ports 0–1023 requires special privileges; these are used by convention by well-known network protocols including HTTP (port 80), HTTPS (443), SMTP email relay (25), DNS (53), and email retrieval (110 POP3, 143 IMAP). Ports 1024–65,535 are typically unprivileged. Clients allocate short-lived ports, often in the range 1024-5000, for their end of TCP connections (leaving ports above 5000 for lesser-known services). An <IP address, port number> pair identifies an IP *socket*. A *TCP connection* connects source and destination sockets. Software accesses sockets via *file descriptors*. UDP has a distinct set of analogous ports, but being connectionless, only destination sockets are used.

TCP HEADER, TCP CONNECTION SET-UP. Figure 10.15 shows a TCP header, including *flag bits*. Prior to data transfer, TCP connection set-up requires three messages with headers but no data. The client originates with a SYN message, i.e., SYN flag set (SYN=1, ACK=0); the server responds with SYN=1, ACK=1 (i.e., both flags set); the client responds with SYN=0, ACK=1. (So ACK=0 only in the initial connection request. In ongoing messages after set-up, the ACK flag is set (ACK=1 in this notation). Thus a firewall may use the criteria ACK=0 to distinguish, and deny, inbound TCP connection requests.) This sequence SYN, SYN-ACK, ACK is the *three-way handshake*; details are given in Section 11.6. Connection termination begins by one end sending a packet with FIN (finish) bit set (FIN=1); the other acknowledges, likewise sends a FIN, and awaits acknowledgement. This is called an *orderly release*. Termination alternatively results from use of the reset flag, RST (*abortive release*).

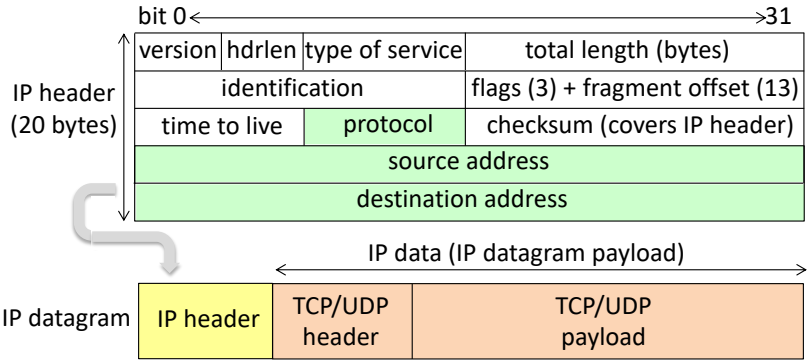

Figure 10.14: IP header and IP datagram carrying TCP or UDP datagram payload. The `identification` field is used in datagram fragmentation and reassembly. TTL (time to live) upper-bounds the number of routers a packet can transit; each router decrements it by one. The IP header checksum is verified (and recalculated, e.g., after TTL updates) at each hop. `Protocol` values of interest are TCP, UDP, ICMP, IP, ESP, AH. `hdrlen` (internal header length, in 32-bit words) is 5 unless an IP `options` field (not shown) is present. The TCP/UDP data part is where Application data is carried, including user data.

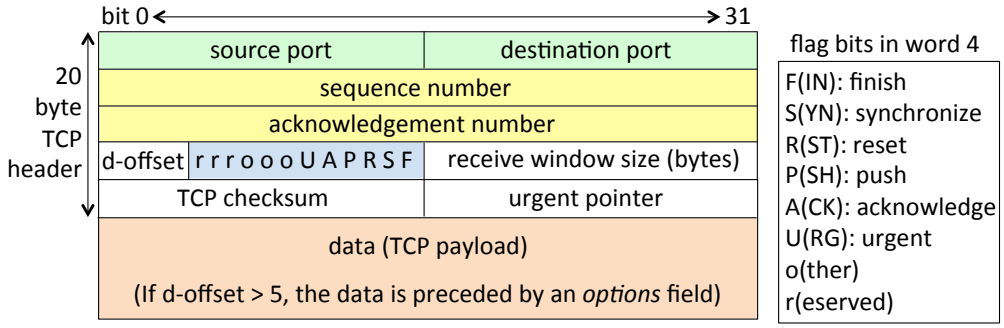

Figure 10.15: TCP header. The 4-bit data offset (`d-offset`) specifies the number of 32-bit words preceding the data. A flag bit of 1 is set (on); 0 is off. Other flag bits beyond our scope are NS (ECN-nonce), CWR (congestion window reduced), and ECE (ECN-echo). The `window_size` advertises how many bytes the receiver will accept, relative to the byte specified in `acknowledgement_number`; values larger than $2^{16} - 1$ bytes are possible by pre-negotiating a window scale option.

ICMP. The Internet Control Message Protocol (ICMP) is an "auxiliary" network-layer protocol at the same level as IP, used to send error, diagnostic, and other control messages from one host (often a router) to another. ICMP messages consist of a body (payload) of an IP datagram (cf. Fig. 10.14) after an IP header, the body a single 32-bit word (8-bit type, 8-bit code, 16-bit checksum; no source or destination ports) plus further content depending on type and code. Example ICMP message types are 3/`destination_unreachable` (a packet cannot be delivered), 5/`redirect` (advising a far-end router to choose a different route), 8/`echo_request` (used by the `ping` utility, to test whether a host can be reached; the receiving host is requested to respond with an

ICMP 0/`echo_reply`), and 11/`time_exceeded` (TTL reached 0). As a basic connectivity test between TCP/IP hosts, `ping` is a standard way to check whether an IP address is populated by an active host; sending ICMP echo requests to a set of addresses is called a *ping sweep*. Firewalls often filter ICMP messages based on their ICMP message type and code fields.

10.7 ‡End notes and further reading

See Zwicky [41] for firewalls, including our dynamic packet filter (FTP) example, full sample firewall rulesets, and architectures; see also Cheswick [8] for technical insights. Chapman [7] gives a 1992 view of challenges with IP packet filters. For augmenting Linux firewall management (`iptables`) with intrusion detection functionality, see Rash [25]. For distributed firewalls, see Ioannidis [15]. For firewall configuration errors, see Wool [33]. Yuan [40] describes a toolkit to analyze firewall configurations. Bejtlich [5, pp. 43–45] relays Ranum's perspective on proxy firewall history and advantages. *Web application firewalls* are application-level firewalls specific to HTTP and HTTPS [32, Ch. 7]. Koblas [20] introduced SOCKS; for version 5, and details on authentication options within it, see RFC 1928 [21].

SSH is due to Ylönen [35]; Snader [27, Ch. 7] provides a highly readable treatment (see also for tunnels, VPNs, and IPsec). SSH2, specified in RFCs 4251-4254 [36, 37, 38, 39], addresses flaws in the original and uses session keys from DH key agreement, replacing RSA. The TOFU model for trust in SSH host keys, positioned in 2006 [36] as a convenient, temporary measure during a "transition period" until a widely deployed trust infrastructure arises, remains widely used. RFC 4255 [26] specifies out-of-band distribution (using DNS) of SSH server key fingerprints. On SSH passwords, for brute-force guessing attacks on servers see Abdou [1]; for timing attacks, see Song [28].

IPsec RFCs respectively cover security architecture [19], IKEv2 [16], AH [17], ESP [18], and mandatory cryptographic algorithms for AH and ESP [34]. For deployment experiences with IPsec, see Aura [4]. For a discussion of deficiencies in IKE key agreement, and proposed replacement JFK, see Aiello [3]. For IKE-related policy management, see Blaze [6]. For a NIST view of VPNs, see SP 800-77 [11]. Foundational Internet specifications include: IP [13], ICMP [24], TCP [14], and UDP [23]. Numerous RFCs discuss aspects of NAT [30, 29]. RFC 5424 [12] standardized `syslog`. Overall, for TCP/IP protocols and *packet fragmentation*, see Stevens [31].

References (Chapter 10)

[1] A. Abdou, D. Barrera, and P. C. van Oorschot. What lies beneath? Analyzing automated SSH bruteforce attacks. In *Tech. and Practice of Passwords—9th Int'l Conf. (PASSWORDS 2015)*, pages 72–91, 2015.

[2] B. Aboba and W. Dixon. RFC 3715: IPsec-Network Address Translation (NAT) Compatibility Requirements, Mar. 2004. Informational.

[3] W. Aiello, S. M. Bellovin, M. Blaze, J. Ioannidis, O. Reingold, R. Canetti, and A. D. Keromytis. Efficient, DoS-resistant, secure key exchange for Internet protocols. In *ACM Comp. & Comm. Security (CCS)*, pages 48–58, 2002. Journal version: *ACM TISSEC*, 2004.

[4] T. Aura, M. Roe, and A. Mohammed. Experiences with host-to-host IPsec. In *Security Protocols Workshop*, 2005. Pages 3–22, and 23–30 for transcript of discussion, in Springer LNCS 4631 (2007).

[5] R. Bejtlich. *Extrusion Detection: Security Monitoring for Internal Intrusions*. Addison-Wesley, 2005.

[6] M. Blaze, J. Ioannidis, and A. D. Keromytis. Trust management for IPsec. In *Netw. Dist. Sys. Security (NDSS)*, 2001. Journal version: *ACM TISSEC*, 2002.

[7] D. B. Chapman. Network (in)security through IP packet filtering. In *Proc. Summer USENIX Technical Conf.*, 1992.

[8] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker (2nd edition)*. Addison-Wesley, 2003. First edition (1994; Cheswick, Bellovin) is free online.

[9] J. A. Donenfeld. WireGuard: Next generation kernel network tunnel. In *Netw. Dist. Sys. Security (NDSS)*, 2017.

[10] B. Dowling and K. G. Paterson. A cryptographic analysis of the WireGuard protocol. In *Applied Cryptography and Network Security (ACNS)*, pages 3–21, 2018.

[11] S. Frankel, K. Kent, R. Lewkowski, A. D. Orebaugh, R. W. Ritchey, and S. R. Sharma. Guide to IPsec VPNs. NIST Special Publication 800-77, National Inst. Standards and Tech., USA, Dec. 2005.

[12] R. Gerhards. RFC 5424: The Syslog Protocol, Mar. 2009. Proposed Standard. Obsoletes RFC 3164.

[13] Information Sciences Institute (USC). RFC 791: Internet Protocol, Sept. 1981. Internet Standard (IP). Updated by RFC 1349, 2474, 6864.

[14] Information Sciences Institute (USC). RFC 793: Transmission Control Protocol, Sept. 1981. Internet Standard (TCP). Updated by RFC 1122, 3168, 6093, 6528.

[15] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith. Implementing a distributed firewall. In *ACM Comp. & Comm. Security (CCS)*, pages 190–199, 2000. See also: S.M. Bellovin, "Distributed firewalls", pages 39–47, *USENIX ;login:* (Nov 1999).

[16] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen. RFC 7296: Internet Key Exchange Protocol Version 2 (IKEv2), Oct. 2014. Internet Standard. Obsoletes RFC 5996 (preceded by 4306; and 2407, 2408, 2409); updated by RFC 7427, 7670, 8247.

[17] S. Kent. RFC 4302: IP Authentication Header, Dec. 2005. Proposed Standard. Obsoletes RFC 2402.

[18] S. Kent. RFC 4303: IP Encapsulating Security Payload (ESP), Dec. 2005. Proposed Standard. Obsoletes RFC 2406.

[19] S. Kent and K. Seo. RFC 4301: Security Architecture for the Internet Protocol, Dec. 2005. Proposed Standard. Obsoletes RFC 2401; updated by RFC 7619.

[20] D. Koblas and M. R. Koblas. SOCKS. In *Proc. Summer USENIX Technical Conf.*, pages 77–83, 1992.

[21] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones. RFC 1928: SOCKS Protocol Version 5, Mar. 1996. Proposed Standard.

[22] L. Phifer. The Trouble with NAT. *Internet Protocol Journal*, 3(4):2–13, 2000.

[23] J. Postel. RFC 768: User Datagram Protocol, Aug. 1980. Internet Standard (UDP).

[24] J. Postel. RFC 792: Internet Control Message Protocol, Sept. 1981. Internet Standard (ICMP). Updated by RFC 950, 4884, 6633, 6918.

[25] M. Rash. *Linux Firewalls: Attack Detection and Response with iptables, psad and fwsnort*. No Starch Press, 2007.

[26] J. Schlyter and W. Griffin. RFC 4255: Using DNS to Securely Publish Secure Shell (SSH) Key Fingerprints, Jan. 2006. Proposed Standard.

[27] J. C. Snader. *VPNs Illustrated: Tunnels, VPNs, and IPsec*. Addison-Wesley, 2005.

[28] D. X. Song, D. A. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on SSH. In *USENIX Security*, 2001.

[29] P. Srisuresh and K. Egevang. RFC 3022: Traditional IP Network Address Translator (Traditional NAT), Jan. 2001. Informational. Obsoletes RFC 1631. See also RFC 2993, 3027 and 4787 (BCP 127).

[30] P. Srisuresh and M. Holdrege. RFC 2663: IP Network Address Translator (NAT) Terminology and Considerations, Aug. 1999. Informational.

[31] W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

[32] R. Trost. *Practical Intrusion Analysis*. Addison-Wesley, 2010.

[33] A. Wool. A quantitative study of firewall configuration errors. *IEEE Computer*, 37(6):62–67, 2004. A 2009 report revisits the study: `https://arxiv.org/abs/0911.1240`.

[34] P. Wouters, D. Migault, J. Mattsson, Y. Nir, and T. Kivinen. RFC 8221: Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH), Oct. 2017. Proposed Standard. Obsoletes RFC 7321, 4835, 4305.

[35] T. Ylönen. SSH—secure login connections over the Internet. In *USENIX Security*, pages 37–42, 1996.

[36] T. Ylönen and C. Lonvick. RFC 4251: The Secure Shell (SSH) Protocol Architecture, Jan. 2006. Proposed Standard. Updated by RFC 8308.

[37] T. Ylönen and C. Lonvick. RFC 4252: The Secure Shell (SSH) Authentication Protocol, Jan. 2006. Proposed Standard. Updated by RFC 8308, 8332.

[38] T. Ylönen and C. Lonvick. RFC 4253: The Secure Shell (SSH) Transport Layer Protocol, Jan. 2006. Proposed Standard. Updated by RFC 6668, 8268, 8308, 8332.

[39] T. Ylönen and C. Lonvick. RFC 4254: The Secure Shell (SSH) Connection Protocol, Jan. 2006. Proposed Standard. Updated by RFC 8308.

[40] L. Yuan, J. Mai, Z. Su, H. Chen, C. Chuah, and P. Mohapatra. FIREMAN: A toolkit for firewall modeling and analysis. In *IEEE Symp. Security and Privacy*, pages 199–213, 2006.

[41] E. D. Zwicky, S. Cooper, and D. B. Chapman. *Building Internet Firewalls (2nd edition)*. O'Reilly, 2000. First edition 1995 (Chapman, Zwicky).