Network Security

Pamela Zave and Jennifer Rexford

December 29, 2018

Please read Appendix A, which is an excerpt from a previous chapter. It is included here to make this chapter self-contained.

Network security can be divided into two major categories, based on where it is provided. *Endpoint security* consists of security measures implemented primarily in endpoints that wish to communicate, and do not trust the rest of the network between them. Endpoint security is always implemented with cryptography. *Infrastructure security* consists of security measures implemented primarily by infrastructure members on behalf of the network's administrative authority, so that the network can provide its specified services. Infrastructure security is usually implemented with packet filtering. The issue of *privacy* is closely related to infrastructure security, because it is often concerned with limiting the power of the infrastructure.

In principle, any security measure might be found in any network in a compositional architecture. In addition to explaining the basics of network security, we will consider how security mechanisms interact with other mechanisms within their networks and across composed networks. Our goal is to understand where security could and should be placed in a compositional network architecture.

1 Security provided by endpoints

Endpoint security is built into session protocols, and implemented in the session endpoints. It uses cryptography to enable two network members to communicate safely, even though they do not trust the rest of the network between them. They assume that part of the network between them might read, absorb, inject, or alter data as it is transmitted. "Communicating safely" can include any of *data confidentiality* (no other entity can read the data), *data integrity* (no other entity can modify the data), *endpoint authentication* (either endpoint is sure of the other endpoint's identity), and *digital signatures* (a digital document was signed by a particular identity).

In this section, §1.1 shows that the identity of an endpoint can come from multiple networks composed by layering. §1.2 introduces cryptographic primitives. §1.3 focuses on single networks in isolation, and what their session protocols can accomplish. In §1.4 we return to multiple networks, considering the

interactions between endpoint security and all the networks in a compositional architecture.

1.1 Trust and identity

Security requirements are based on which network members do and do not "trust" each other. Of course a network member is a software or hardware module; it cannot trust in any ordinary sense of the word, and has no legal responsibility that it can be trusted to fulfill. For the purpose of establishing trust, a network member that is an endpoint of a session has an *identity*. This identity is given to the other endpoint of the session in answer to the question, "Whom am I talking to?"

This role implies that an identity should have meaning in the world outside the network. Often it is closely associated with a legal person—a person or organization—who is legally responsible for the network member. The identity is usually the source of the data that the network member sends during the session.

To understand where identities come from, consider the Web session pictured in Figure 1. At the lower level, a TCP session traverses a chain of bridged IP networks. At the upper level, a distributed system is viewed as a network, which is always possible even though their structures as networks are usually too simple to bother with. In this Web-based application network there is a dynamic link (implemented by the TCP session) between a browser and a server, on which an HTTP session is taking place. Placed above the client's browser there is a user whose clicks and keystrokes provide input to the browser.

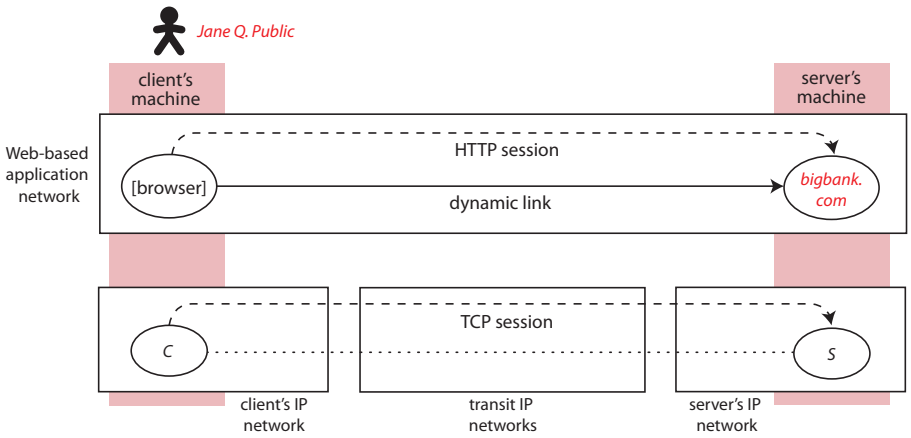

Figure 1: Identities in a Web application.

In this example, the server's machine has two interfaces to two network members, each with a name in the namespace of its network. In its IP network

it has IP address S. In the Web network it has domain name *bigbank.com*. The client's machine also has two network members, but the browser does not really have a name in the Web-based application network, because it only initiates sessions and never receives them. However, the user of the browser is a person named *Jane Q. Public*, and we can imagine her as a member of an even-higher-level distributed financial system.

If the two endpoints of the TCP session need to authenticate each other (as they should, for a banking transaction), what identities do they give as their own? The general answer is that each gives its network name or the name of a higher-level network member that is using it. Either IP interface could give its IP name, but it would not be a very good identifier—too transient, or with too little meaning in the outside world. The server's IP interface S would send its Web name *bigbank.com*. The browser's IP interface C would send its user's name *Jane Q. Public*.

For endpoint authentication, a member must have access to a secret associated with the identity it provides. One kind of secret, useful when the two endpoints have an ongoing relationship, is a password. The server *bigbank.com* knows Jane's password, and she can type it into the browser when requested.

For the important endpoint security protocols, however, the secret is always a public/private cryptographic key pair (semantics given in the next section). The relationships among the important entities are shown in Figure 2. The identity is responsible for the packets sent by the network member, and the network member has access to the public key and its paired private key.

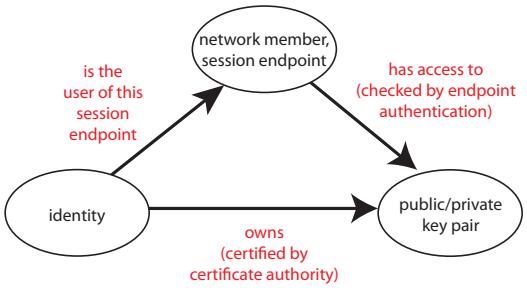

Figure 2: Relationships among entities in endpoint security.

A "certificate authority" is trusted to ascertain that a particular public key belongs to a particular identity; it issues a certificate to that effect and signs it digitally. Thus when an endpoint receives a certificate, it can trust the identity that goes with the key (at least, as well as it trusts the certificate authority). As indicated above, identities found in certificates include names of legal persons, domain names, and IP addresses.

It should be noted that trust between communicating endpoints is not necessarily simple or absolute. One or both endpoints may wish to remain anony-

mous. One endpoint can delegate its identity to another, as in Content Delivery Networks (see sidebar). Two endpoints may be communicating to negotiate a contract, and (because they do not trust each other completely) need to communicate through a third party trusted by both. A trusted broker can ensure, for example, that both parties sign the exact same contract [6].

1.2 Cryptographic primitives

This chapter provides a mere sketch of cryptography, with just enough information for readers to understand how it is being used. For a more substantial but still easy-to-read explanation of cryptography in network security, see Kurose and Ross [26].

In *public-key cryptography*, an identity generates and owns a coordinated pair of keys, one public and one kept private and secret. The important properties of these keys are that (i) it is extremely difficult to compute the private key from the public key, and (ii) plaintext encrypted with the public key can be decrypted with the private key, and vice-versa.

A simple challenge protocol is sufficient to determine that an endpoint has access to a public/private key pair. Suppose that an endpoint B receives packets from another endpoint A that claims to have public key K^+. B can make sure of this by sending a *nonce* (a random number used only once in its context) n. A is supposed to reply with $K^-(n)$, which is n encrypted using the private key K^- that goes with public key K^+. B then decrypts the reply with K^+. If the result is n, then B has authenticated that the other endpoint indeed has access to public key K^+ and its private key K^-.

Any member can send confidential data to A by encrypting the data with K^+. Public-key cryptography is also used to create and check digital signatures.

Public-key cryptography is computationally expensive, so it is used only to encrypt small amounts of data. For larger data streams, the more efficient *symmetric-key cryptography* is used. Symmetric-key cryptography requires that both endpoints have the same secret key, which is used both to encrypt and decrypt the data. Distributing shared, secret keys is another use for public-key cryptography.

The other cryptographic primitive mentioned in this chapter is a *cryptographic hash.* A cryptographic hash is computed by a function H from a digital message m (of any length) to a fixed-length bit string. Its important property is that, given a hash $H(m)$, it is extremely difficult to compute a different message m' such that $H(m) = H(m')$.

1.3 Session protocols for endpoint security

This section has two purposes, the first of which is to explain the four elements of endpoint security. The second purpose is to acquaint you with the two most important protocols for endpoint security, TLS and IPsec. Transport Layer Security (TLS) is the successor to Secure Sockets Layer, and is usually considered to be an extension of TCP. "IPsec" refers to a family of related IP

protocols, comprising the Authentication Header and Encapsulating Security Payload (ESP) protocols, each of which can be used in "transport mode" or "tunnel mode." ESP is more useful than Authentication Header, so only ESP will be discussed here.

The first subsection defines the four elements of endpoint security, after which we give details on how they are implemented in TLS and ESP.

1.3.1 Elements of endpoint security

The first element of endpoint security is endpoint authentication. This is a handshake in which either endpoint can send its identity and public key to the other endpoint, and the other endpoint can challenge to verify it.

The second element of endpoint security is *key exchange*. Using public-key cryptography for security, the endpoints agree on shared (symmetric) keys. Typically there are different keys for data encryption and for message authentication. Also typically, a different key is used for each direction of transmission.

The third element of endpoint security is *data encryption*. Each endpoint sends data encrypted with a key, and the other endpoint decrypts it with the same key. According to the mathematics of cryptography, encryption satisfies the requirement of data confidentiality.

The fourth element of endpoint security is *message authentication*, to satisfy the requirement of data integrity. Each packet is sent with a "message authentication code" computed in a secret way (using a shared key) from its contents. If an attacker changes or inserts packets while they are being transmitted, it will not be able to compute correct authentication codes for the packets, and the discrepancy will be detected by the receiver.

Please read Appendix B, which is an excerpt from a previous chapter. It is included here to make this chapter self-contained.

1.3.2 Endpoint authentication

TLS is a security protocol composed with (embedded in) TCP. If the URL of a Web site begins with `https://`, then its clients should make requests of it using IP protocol TCP and destination port 443, signifying the use of TLS embedded in TCP. The client and server first have a TCP SYN handshake, after which they begin the TLS handshake as shown in Figure 3.

The first three messages of the TLS handshake contain a version of the challenge protocol in §1.2, where the client is B and the server is A. The first message from A to B contains its certificate, from which B can get its public key. The client should validate the certificate in various ways, including checking that its identity is the one requested, checking that it has not expired, and checking that it has been issued by a legitimate certificate authority.

The "pre-master secret" in the handshake acts as B's nonce in the authentication protocol. Here the roles of public and private keys above are reversed, as B *encrypts* the pre-master secret with K^+ and sends it to A to be *decrypted.*

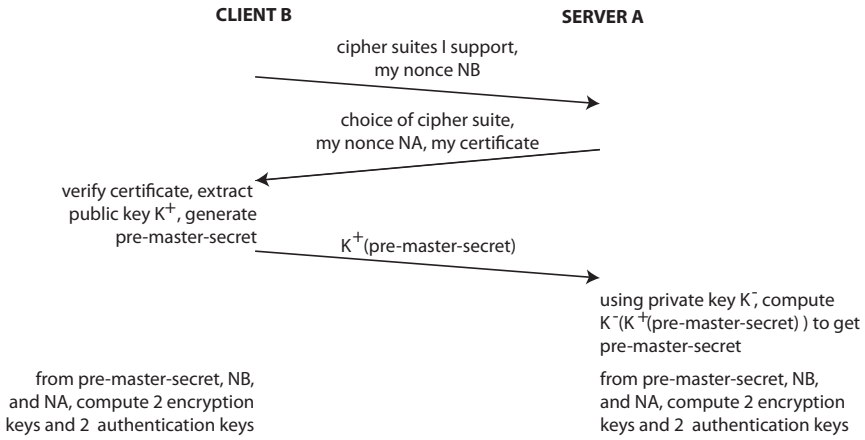

Figure 3: Important messages of the TLS 1.2 handshake.

Although B does not know directly whether A gets the same value of the pre-master secret, the remainder of the protocol and session will not work unless both A and B have the same value of it.

Endpoint authentication in ESP also uses public/private keys and certificates. ESP endpoint authentication is optional and less common, because ESP is meant for use by all IP endpoints, most of which have no certificates, and many of which have no identities other than transient IP addresses.

To give a third perspective on endpoint authentication, the Accountable Internet [2] (see sidebar) is a proposal based on the principle that Internet names should be the persistent identities of Internet members, and that they should be "self-certifying." This means that any other member communicating with a member can authenticate its name, even without trusting a certificate authority. Clearly this could be achieved if the name of a member were its public key, but public keys (requiring at least 2000 bits to be adequately secure) are too long for network names. The Accountable Internet solves this problem by using as a member's name a 144-bit cryptographic hash of its public key.

1.3.3 Key exchange

Data encryption uses symmetric-key cryptography, and message authentication is based on cryptographic hashes. Both tasks require shared secrets, *i.e.,* both endpoints must have the same secret keys. "Key exchange" is the task of establishing shared secrets in session endpoints.

So we see that endpoint security can require a combination of several cryptographic tasks, and for each of these tasks there are many possible algorithms (counting all variations of a few basic algorithms). A "cipher suite" is a collection of algorithms and parameter choices for doing all the crytographic tasks

within a security protocol. Both TLS and ESP allow endpoints to choose the cipher suite they will use.

Returning to Figure 3, the first message of the TLS handshake, from client to server, includes a list of the cipher suites that the client supports. The server's reply chooses from among the suites offered. The handshake also implements TLS key exchange. Both endpoints generate secret keys in the same way from the same raw materials, which are the pre-master secret and two nonces. Although the nonces are transmitted as plaintext, the pre-master secret is encrypted, so no one but the server can decrypt it. From these three bit strings, both endpoints use the same algorithm to compute a "master secret." Both endpoints then slice up the master secret in the same way to get multiple keys for handling data. To complete TLS initialization, both endpoints send the other a message authentication code for all the handshake messages.

ESP endpoints do key exchange by means of the Internet Key Exchange (IKE) protocol, which is different from the TLS handshake but also uses public-key cryptography to distribute secrets safely. The result is that each ESP endpoint has a long record including choice of cipher suite and actual keys. Use of IKE to set up an ESP session is not always necessary because these records can also be introduced into ESP endpoints by configuration.

1.3.4 Data encryption

TLS relies on TCP to deliver a reliable, ordered, duplicate-free byte stream. The byte stream consists of TLS messages, some of which are control messages carrying out the initial TLS handshake and closing the session at the end. All the TLS messages in between these control messages carry encrypted data. Note that neither the TCP headers on packets nor the small TLS headers (message type, TLS version, length) are encrypted. Encryption uses two symmetric keys, each one used to encrypt and decrypt data in one of the two directions.

In ESP, all of the control data needed for a session is included in one of the long records that are pre-configured or created by IKE. A 32-bit "security parameters index" points to a stored record, and is part of each ESP header. Just as TLS has a different encryption key for each direction, ESP headers in each direction carry different security parameters indices.

ESP can be used in two modes, "tunnel mode" and "transport mode." Tunnel mode is a straightforward implementation of layering; a complete packet (usually an IP packet) is the encrypted payload of the IP/ESP packet. In tunnel mode, an IP/ESP session provides to an overlay network a link with data confidentiality and integrity.

In transport mode, on the other hand, some other IP session protocol—usually TCP—is embedded in ESP. So ESP and TCP are being composed in ESP transport mode, just as TLS and TCP are composed in the session protocol known as "TLS." It is interesting to note that in the two cases the embedding goes the the opposite ways: TLS embeds security inside TCP, while ESP transport mode embeds TCP inside security.

1.3.5 Message authentication

Data integrity means that the data stream received is the same as the data stream sent, with no tampering during transmission. Data integrity is ensured by the process called "message authentication." Message authentication uses a cryptographic hash function (§1.2) and a shared secret called an "authentication key." Taking packet data d, the process computes a "message authentication code" (MAC) by appending to the data the authentication key k, and then applying the hash function to $d+k$. The MAC $H(d+k)$ is then appended to the data in the packet. The receiver of the packet can make the same computation; if its result agrees with the MAC, then the data has not been tampered with.

In TLS, every record has a MAC for authentication, and there is a different authentication key for each direction. Assuming one TLS message per packet, the MAC as described above can show that a received packet came from sender. But this algorithm has the limitation that an attacker with access to the packet stream can still delete, re-order, or replay (insert a packet that has already been delivered) packets. (Note that TCP checking of byte sequence numbers is no protection against these attacks, because the TCP headers are not encrypted and the attacker can alter them.) The TLS solution is for each endpoint to keep track of message sequence numbers as TLS messages are sent and received. The sequence number is not transmitted directly in the message data, but it is included in the bit string hashed to compute the MAC. For a packet to be accepted, the receiver must be re-computing its MAC with the same sequence number that the sender used. TLS records cannot be received out-of-order because of the guarantees provided by TCP, on which it depends.

ESP authentication appends a MAC to each packet, whether in tunnel or transport mode. The authentication key for each direction is part of the relevant security record.

Authentication in ESP is concerned with packet replay, to the exclusion of deletion or re-ordering, because replay is the most serious security vulnerability. ESP headers contain explicit packet sequence numbers, which are included in the data on which the MAC is computed. ESP does not have TCP to depend on, so packets could arrive out-of-order, and the receiver cannot predict the exact sequence number of the next packet. So ESP checks only for received packets with sequence numbers that have already been received (and deletes them) [22]. This is sufficient to defend against replay attacks, because an attacker cannot change the sequence number of a packet it replays.

1.3.6 Uses of endpoint security

Properties of the protocols are summarized in Figure 4.

Use of TLS for Web traffic has been growing steadily, and now exceeds the amount using TCP. TLS is also widely used by other application protocols. ESP is most commonly used to make Virtual Private Networks (see sidebar).

Not surprisingly, developers building applications on UDP are also interested in endpoint security. For UDP transmission, there is a security protocol called

	ENDPOINT AUTHENTICATION using public/private keys and certificates	KEY EXCHANGE	DATA ENCRYPTION using a different symmetric key in each direction	MESSAGE AUTHENTICATION using a different authentication key in each direction
TLS 1.2	optional for initiator, mandatory for acceptor	performed by TLS handshake	IP, TCP, and TLS headers unencrypted	defends against all tampering, including replay, deletion, and re-ordering
ESP tunnel mode	optional for both endpoints	performed using separate IKE protocol, or unnecessary because pre-configured	IP and ESP headers unencrypted, encapsulated IP packet encrypted	defends against replay attacks, not deletion or re-ordering
ESP transport mode			IP and ESP headers unencrypted, TCP header encrypted	

Figure 4: Summary of protocol properties.

DTLS (Datagram Transport Layer Security) that is as similar as possible to TLS. DTLS introduces the notion that a sequence of UDP packets go together in a session, which is not present in plain UDP. It should be clear from the previous sections that, because DTLS is not embedded in TCP, its designers had to solve two problems: (i) the TLS handshake assumes reliable delivery of the handshake messages, and (ii) DTLS message authentication cannot rely on the property that packets are delivered reliably, in order, and duplicate-free, so that packet sequence numbers can be computed independently at each endpoint. DTLS solves the first problem by incorporating packet-loss detection and retransmission into the DTLS handshake. DTLS solves the second problem by using explicit sequence numbers, exactly as ESP does.

Although endpoint security is well-established in today's Internet, attackers always get better, and the war is not won. A report on the known attacks on TLS 1.2 [33] cites a wide variety of security threats. There is continual exploitation of problems in cipher suites and their implementations, but also continual improvements to them in response. There are also problems with certificate technology, theft of private keys, and vulnerabilities concerning application programming. Certificates are often validated poorly or not at all [13].

Recently a standard for TLS 1.3 has been approved. The new standard requires use of an up-to-date cipher suite. It also incorporates optimizations so that typical TCP/TLS setup times are reduced from three round-trip times (RTTs) to two.

1.4 Compositional endpoint security

This section is concerned with the interactions between endpoint security and other aspects of networking. As a consequence, we will be looking at multiple

composed networks as well as protocols within a single network.

1.4.1 Performance

Data encryption and message authentication increase required bandwidth and computational resources slightly, but no one seems to regard these as problems. The significant, direct performance costs of endpoint security are incurred by endpoint authentication and key exchange, which consume significant compute resources, as well as increasing latency. For example, the TLS 1.2 handshake consumes two RTTs, added to the one for TCP. Even with short RTTs, a small fraction of TLS 1.2 setups take 300 ms or more [30], due to increased computation time.

TLS handshake overhead is enough to make Web servers more vulnerable to denial-of-service (DoS) attacks, as attackers can create a surge of new TLS session requests. Attempts to optimize handshakes by caching and sharing secrets among sessions have created new security vulnerabilities in TLS 1.2 [33].

The performance issue is much more serious in applications for the Internet of Things (IoT), because these applications tend to have periodic or irregular short communications from a large number of networked devices to centralized analysis or publish/subscribe servers. Message Queuing Telemetry Transport, a protocol for IoT applications, is well-designed from this perspective, because many brief application messages can share the same long-term TLS session. Even so, group events (such as initialization of a fleet of vehicles) can easily create spikes in the load on centralized servers [16].

1.4.2 Session-protocol composition

The most significant issue for composition of endpoint-security protocols is their relationship to TCP, because TCP does so many things: congestion control, reliability, and packet ordering. We have seen that TLS depends on being embedded in TCP, while DTLS and ESP do not. This should not be a problem, unless real or perceived implementation constraints cause designers to make bad choices. For example, some VPN designs use TLS to implement secure links in the underlay networks (compare this to Figure 16). Because of the dependence of TLS on TCP, this design is layering one instance of TCP over another instance of TCP! If that sounds like a bad idea, it is [17]. The obvious problem, called "TCP meltdown," is as follows.

TCP provides reliability by detecting lost packets by means of a timer, and requesting retransmission of a packet when it does not arrive in time. For each session, TCP sets the timeout interval independently and adaptively. It can happen that the timeout interval on the upper-level instance of TCP becomes shorter than the timeout interval on the lower-level instance. In this case the lower-level session is experiencing reduced throughput, because it is waiting a longer time for each packet. At the same time, the upper-level session is having frequent timeouts, making frequent requests for retransmission, and therefore

demanding increased throughput. This mismatch drastically degrades the end-to-end throughput.

Endpoint-security protocols can also incur problems simply because they are not well-enough supported by their network designs. Unfortunately, this is the case for ESP in IP networks. Networks need session identification, and session identifiers should be standard parts of network headers. Yet IP groups session identifiers with protocol headers, and they are not standardized across all protocols. The first 32 bits of a TCP header consists of two port numbers, which distinguish the session's packets from all others with the same source and destination names, and which can easily be matched with the port numbers of reverse packets of the same session. In ESP, on the other hand, the first 32 bits of the protocol header are a security parameters index (§1.3.4), which is completely different in the forward and reverse directions, and cannot be used to associate them. The consequence is that a stateful firewall or NAT box at the edge of a private network, configured to only allow two-way sessions initiated in the outgoing direction, will not allow ESP sessions.

In this case protocol composition enables a workaround to the problem. ESP, whether in tunnel or transport mode, is embedded in UDP with well-known port 4500. (A well-known port for UDP + ESP composition is necessary because UDP headers have no place for the "next header," as IP and ESP headers do.) In this way a two-way sequence of UDP packets forms an identifiable session, and the stateful firewall or NAT box does not see the ESP headers at all.

1.4.3 Middleboxes

Please read Appendix C, which is an excerpt from a previous chapter. It is included here to make this chapter self-contained.

There is a profound interaction between middleboxes and endpoint security in a network. From the perspective of middleboxes, endpoint security can prevent them from doing their jobs by making packets unreadable. From the perspective of endpoint security, a session with middleboxes, even if they are benign and correct, may not have the property of data integrity that the cryptographic algorithms are designed to verify. A middlebox might benignly and correctly tamper with the end-to-end packet stream, for example by converting the application protocol or data representation. Furthermore, endpoint authentication is complicated by the presence of middleboxes, which may hide the endpoints from each other, and which may themselves require authentication.

The conflict between "deep packet inspection" (reading the payloads of packets) and encryption has no known mitigation when the relationship between endpoints and middleboxes is adversarial. The prototypical example is middleboxes for law enforcement, called "lawful intercept." In the absence of technical solutions, lawful intercept is the subject of contentious social and legal discussions.

When some form of cooperation between endpoints and middleboxes is possible, on the other hand, there are several approaches to the problem. Cooperation is often a reasonable expectation. Many middleboxes are directly beneficial

to endpoints (caches reduce Web page load times, data compression minimizes the data usage of mobile devices, virus scanners protect user machines). Other middleboxes protect network infrastructure, benefiting all well-behaved users indirectly. Some service providers may be offering cheaper service as an incentive for users to accept advertising or data collection. And, of course, in private networks all endpoints and middleboxes may belong to the same administrative authority. In the remainder of this section we will present three approaches to cooperatively combining endpoint security with middleboxes.

The simplest way to combine encryption and middleboxes is to layer the middleboxes above encryption, as illustrated by Session Initiation Protocol (SIP) networks for controlling multi-media applications (a sidebar will be forthcoming). The SIP network is an application-specific network with a very particular design in which all middleboxes are publicly-known proxies. Each of the initiating and accepting endpoints of a session always has a proxy performing certain functions for it, so that each SIP session is compounded of three simple sessions (see Figure 5). The members of the SIP network, including endpoints and proxies, all have human-friendly names. In this context endpoint authentication between adjacent elements of a compound session is meaningful and sufficient—by design, proxies are trusted, and trusted to authenticate the next element in the chain of a complex session.

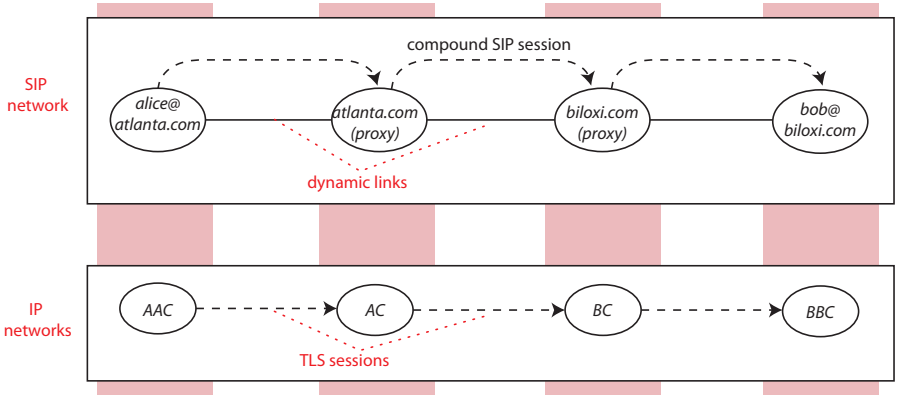

Figure 5: SIP middleboxes (proxies) are layered above encryption.

In the SIP session of Figure 5, SIP messages are routed through proxies to *bob@biloxi.com* along virtual links that may be pre-existing, or may be created at the time they are needed. Each link is implemented by a TCP or TLS session through one or more IP networks. *alice@atlanta.com* can be configured with the IP name *AC* of its proxy. *atlanta.com* can look up the IP name of *biloxi.com* in DNS. It is the business of the proxy *biloxi.com* to know the current IP name of *bob@biloxi.com*, because the endpoint is registered with the proxy. For TLS endpoint authentication of an IP member, its identity is the name of the SIP

member *(e.g., bob@biloxi.com)* that is attached to the IP member *(e.g., BBC)*.

The other two approaches work entirely within a single network, and are based on composition and augmentation of session protocols. They are more general than the SIP approach, in the sense that middleboxes can be used more fluidly and privately than in SIP.

The second approach allows middleboxes of any type, provided that they are promoted to being proxies. This approach is best represented by Middlebox TLS (mbTLS) [31], in which the middleboxes create a compound session consisting of an end-to-end chain of simple TCP sessions. Along the chain from initiator to acceptor, there is first a set of middleboxes inserted on behalf of the initiator, followed by a set inserted on behalf of the acceptor (see Figure 6). A particular middlebox can be inserted into the chain either because packets are forwarded to it, or because the TCP initiator of this simple session has chosen its name as destination of the packets. In the latter case, the TCP initiator may receive this name in a list that comes from the compound-session initiator, or may receive this name from a DNS lookup of a domain name.

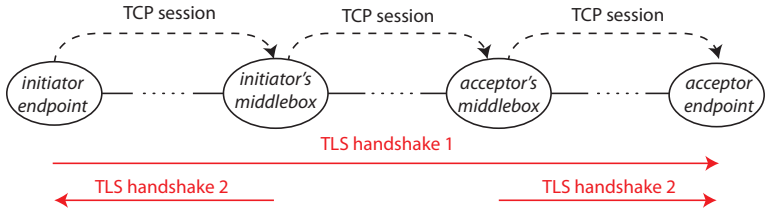

Figure 6: Control signaling to set up an mbTLS session.

Within the end-to-end chain of TCP sessions, the initiator and acceptor first have a normal end-to-end TLS handshake for endpoint authentication and key exchange. Then each middlebox initiates a secondary TLS handshake with the next element in the direction of its sponsoring endpoint. For example, if there are two middleboxes *M1* and *M2* inserted on behalf of the initiator, *M2* initiates a secondary handshake with *M1*, and *M1* with the initiator. The secondary handshakes exchange symmetric and authentication keys for the individual simple sessions. They can also perform endpoint authentication of middlebox identity (in this case the responsible owner), software version and configuration, security properties of the hardware/software platform, etc. This makes sense because the middleboxes associated with each endpoint are working in cooperation with it, even if they have an adversarial relationship with the middleboxes of the other endpoint.

After the secondary TLS handshakes, data is transmitted, with separate encryption and message authentication in each simple session. The middlebox members shown in the figure are the IP interfaces of the middleboxes, and they decrypt the transmitted data before delivering it to the middlebox application code. Note that the "midpoint" simple session between initiator and acceptor

middleboxes has no secondary handshake; in this simple session, the keys chosen by the primary TLS handshake are used. An earlier version, Multi-Context TLS [32], allowed the endpoints to place constraints on the read/write access of middleboxes.

The third approach is based on new results in cryptography. At one extreme, fully homomorphic encryption [14] makes it possible to compute any function on encrypted data without learning more about the data than the function's value. Although fully homomorphic encryption is currently impractical (it is too expensive computationally, by orders of magnitude), there are less capable algorithms for computing functions on encrypted data with performance that may be feasible for current use.

BlindBox [34] is a proposal for allowing middleboxes to operate on encrypted data. BlindBox middleboxes can apply detection rules of the kind commonly used by virus scanners, intrusion-detection systems, and parental filters. These rules often identify keywords and other exact strings at particular positions in the data, which the middleboxes can search for. The scheme also allows a middlebox that has found a keyword or other suspicious string, as probable cause of a security violation, to decrypt the entire packet.

BlindBox is implemented as an extension of TLS. In addition to the basic TLS handshake, endpoints must generate extra keys. The data must be sent end-to-end twice (redundantly), once in the ordinary TLS form and once in a reformatted and re-encrypted form suitable for the Blindbox algorithms.

The biggest overhead incurred by Blindbox is due to rule preparation, because the middlebox must have the rules themselves encrypted with a session-dependent key. The endpoints must not know the rules (this would make them easier to evade) and the middlebox must not know the key (otherwise the guarantee of confidentiality would be lost). So who can encrypt the rules? For every keyword in every rule, the endpoints must generate and transmit to the middlebox a special encryption function that incorporates yet obfuscates the session-dependent key. The middlebox must then check the two for agreement (in case one of the endpoints is insecure) and then apply the encryption function to the rule. This results in very high performance overhead, which means that Blindbox is currently practical only for long-lived sessions or small rule sets.

Although both Middlebox TLS and Blindbox are very promising efforts, it seems clear that their complexity and non-uniform communication among participants are weaknesses. Hopefully further research in this area will bring more uniformity and better-understood control mechanisms. Complexity itself is a security vulnerability, because it provides a larger "attack surface" for adversaries to probe.

1.4.4 Mobility

In its strongest sense, mobility enables a session to persist even though the network attachment of a device at its endpoint is changing. Currently implementations of true mobility for Internet endpoints are rare, for reasons explained in the chapter on *Mobility*. However, even if Internet endpoints could be mobile,

there would probably be no interaction between mobility and endpoint security because the authenticated identity of the endpoint would be at a higher level than its network attachment. For example, the identity of a Web server is its domain name, not its IP address.

On the other hand, thinking about network mobility brings up a possibility we might call "reverse mobility"—where the higher-level identity changes while the lower level remains the same. It turns out that this is a real issue. For example, after *Jane Q. Public* enters her password (§1.1), she might walk away from her machine, and then any other person who walks by could retrieve her personal data and request transactions on her bank account. For this reason, secure distributed applications require periodic re-authentications of the identity of the person using them.

1.4.5 Architectural summary

With respect to architectural consequences, endpoint security has two major aspects: session setup (endpoint authentication and key exchange) and data transmission (data encryption and message authentication). Data transmission will be discussed first.

The section above has presented some nascent efforts to make middleboxes and data encryption compatible. They may mature into easily usable technology, but until then, it is better to avoid the conflict if possible. There are three ways to keep data encryption and middleboxes out of each other's territory:

- separate encryption and middleboxes along the length of a session;
- put the middleboxes in a layer above encryption, so they are encryption endpoints and see plaintext;
- put the middleboxes in a layer far enough below encryption.

The first technique is used by VPNs. In Figure 16, the path between *V1.4* and *V2.8* lies completely within the enterprise's dedicated network (assuming a dedicated link between sites). On this path, the enterprise can place middleboxes, which will see all packets in unencrypted form. In this design the VPN server is taking over the compute server's job as security endpoint, which has the additional advantage of reducing the incidental load on the compute server.

The second technique is illustrated by the SIP multi-media network. In the SIP network, proxies are trusted first-class members that see all packets unencrypted. Endpoint security is implemented at a lower level, encrypting packets only when they are traveling on links of the SIP network between proxies and endpoints.

The third technique is illustrated by the packet in Figure 7, which is a simplified version of Figure 13. The packet is transmitted through an architecture with four layered networks from *N1* (lowest) to *N4* (highest). In this architecture encryption is a service of *N3*, so that the *N4* portion of the packet is encrypted. However, the headers at levels *N1* through *N3* are in plaintext, and

can be examined or manipulated by any middlebox on the path. We know that *N2* is a cellular network, which has middleboxes. If these middleboxes are interested only in *N2* and *N3* headers, then encryption at a higher level is no inconvenience to them. Similarly, middleboxes in *N3* that only look at headers have no problem with the encryption.

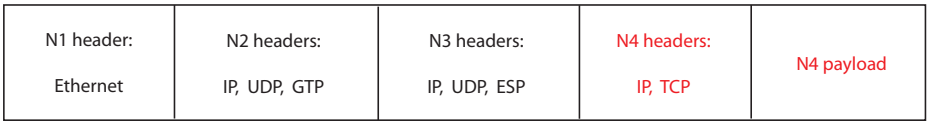

Figure 7: A packet transmitted through an architecture of four layered networks. Only the portions in red are encrypted.

There are, of course, other constraints on layered architectures. In this chapter we have seen an important one, that TCP should not be layered over TCP. Another constraint drawn from endpoint security is that middleboxes operating on encrypted data require the upper-level architecture hidden by encryption to be fixed. For example, if there are Blindbox middleboxes in *N3*, then there cannot be another layer inserted for other purposes between *N3* and *N4*. If there were, Blindbox's search for specific keywords at specific positions in the encrypted data would fail.

Turning to the subject of session setup, cost in terms of latency and computation is an issue in some cases. Technology for endpoint security is always becoming more efficient, but the delay of extra round trips and computation for setup cannot be removed. So the obvious optimization is longer-lived secure sessions, with keep-alive signals and regular key changes as built-in services (note that TCP, TLS, and DTLS now have keep-alive options). Within these long-lived sessions, many shorter sessions for bursty communication can be nested. This strategy can be implemented using layering, so the long-lived secure sessions are in an underlay network, and they implement persistent links in an overlay network. Alternatively, the strategy can be implemented in one network, by embedding the bursty application protocol in the secure session protocol.

Of all aspects of endpoint security, endpoint authentication is by far the most varied and flexible. Normally the identity of a network member is its name, or the name of a higher-level member on the same machine that is using this session (§1.1). A higher-level identity is more recognizable and persistent in the world outside the network. A lower-level identity such as the network name of the endpoint can be temporary, and can change during the session because of mobility. On the other hand, it can be verified more directly. If the identity of an endpoint is its network name, then packets from the other endpoint will not reach it unless it has given its true identity.

Because identity is embodied in data structures such as certificates, pub-

lic/private key pairs, and digital signatures, it can also be moved around. Identity can be delegated, as a Web site of origin delegates its identity to a content-delivery server by sharing its certificate and keys. At the extreme, if the only reason for communicating is access to data, then identity can be attached to the data itself instead of to the endpoint that provides it. This is done by attaching a digital signature, as in Named Data Networking (sidebar forthcoming).

2 Security provided by infrastructure

The administrative authority (AA) of a network is responsible for protecting infrastructure members and well-behaved user members from attackers. It is also responsible for providing network services as specified. The infrastructure members are controlled by the AA and trusted by it to perform security tasks with this goal.

This section begins with an overview of the diverse goals of infrastructure security (§2.1). Regardless of the goal, most infrastructure security is provided by some kind of packet filtering, so in the bulk of the section §2.2 covers packet filtering and §2.3 covers security mechanisms that are not packet filtering. In §2.4 we return to the subject of composition, considering the interactions among security mechanisms, composition operators, and other aspects of networking.

2.1 Goals of infrastructure security

For infrastructure security, the goals are complex, as they are contingent on threat models as well as many assumptions and interests. This section divides goals into two major types.

2.1.1 Preventing or mitigating resource attacks

A *resource attack* seeks to make its victim unavailable by exhausting its resources. In networking, resource attacks are usually called *flooding attacks,* because they entail sending floods of packets toward the victim. Flooding attacks are one type of *denial of service (DoS) attack*.

The intended victims of flooding attacks vary. If the victim is a public server or other endpoint, the attack might seek to exhaust its compute-cycle or memory resources. An attacker might also target some portion of a network, seeking to exhaust the bandwidth of its links. A bandwidth attack can make particular endpoints unreachable, and can also deny network service to many other users whose packets pass through the congested portion of the network. Note that some public servers such as DNS servers are part of the infrastructure of a network, so a flooding attack on a DNS server is an attempt to deny network service to a large number of users.

If an attacker simply sends as many packets as it can toward a victim, the resources expended by the attacker may be similar to the resources expended by the victim! For this reason, an effective flooding attack always employs some

form of *amplification,* in which the attacker's resources are amplified to cause the victim to expend far more resources. Here are some well-known forms of amplification:

- A "botnet" (see sidebar) is formed by penetrating large numbers (as in millions) of innocent-but-buggy Internet members, and installing in them a particular kind of malware. Subsequently the attacker sends a triggering packet to each member of the botnet, causing it to launch a security attack unbeknownst to the machine's owner. A flooding attack from many network members, particularly members of a botnet, is called a "distributed DoS attack."

- An "asymmetric attack" sends requests to a server that require it to expend significant compute or storage resources for each request, so that a relatively small amount of traffic is sufficient to launch a significant attack. A typical IP example is a "SYN flood," in which the victim receives a flood of TCP SYN packets. Each packet causes the server to do significant work and allocate significant resources such as buffer space. Also in IP networks, attackers can flood DNS servers with random queries (a "random subdomain attack"). These will force the servers to make many more queries, because they will have no cached results to match them. In a Web-based application network, the attacker can send particular HTTP requests that force the Web server to do a large amount of computation.

- An attacker can send many request packets to public servers, with the intended victim's name as source name. This "reflection attack" causes all the servers to send their responses to the victim. It amplifies work because responses (received by the victim) are typically much longer than requests (sent by the attacker).

- In an Ethernet network, a forwarder's response to receiving a packet to an unknown name is to broadcast it across the network. An attacker can amplify any packet by broadcast, simply by putting in an unused destination name.

If network infrastructure discovers where attack traffic is coming from, it can often block traffic from the attacker to stop the attack (or, eventually, take legal action). For this reason, attackers employ various techniques to hide themselves, for example:

- In an IP network, a sender can simply put a false source name in the packet header. This is necessary for reflection attacks. It cannot be used if the attack entails a dialogue with the victim, because a false ("spoofed") source name would prevent the dialogue.

- With a botnet, none of the bots sending attack traffic are actually responsible for the attack. Even if bots use true source names, there may be too many of them to cut off.

- An attacker can hide by putting a smaller-than-usual number in IP packets' "time-to-live" fields, so that the packets disappear after they have done their damage in congesting the network, but before they reach a place where defenses are deployed.

The examples of amplifications and hiding techniques show that flooding attacks are network-dependent, because they exploit vulnerabilities in the protocols of specific networks. Nevertheless, their effects are *not* network-dependent, because of "fate sharing." All the network members and applications on a machine share the same physical resources and physical network link, so if resource exhaustion causes a machine to crash, thrash, or become disconnected, all programs running on the machine will share the same fate.

Flooding attacks are a very serious problem in today's Internet. There are businesses that generate them for small fees. They target popular Web sites and (especially) DNS [12]. The worst attacks are mounted by enterprises, albeit illegal ones, that can draw on the same kind of professional knowledge, human resources, and computer resources that legitimate businesses and governments have. Such attackers will use many attacks and combinations of attacks at once, and can continue them over a long period of time.

2.1.2 Blocking specific communications

Obviously, the default behavior of a network is to provide all communication services requested of it. These services should be provided according to explicit or implicit agreements about quality, privacy, and billing. There are, however, specific communications that a network treats differently and prejudicially. These communications are prevented, secretly recorded, or tampered with in some other way.

Here are some well-known examples of specific communications that may be prevented using the mechanisms of infrastructure security:

- Email spam and voice-over-IP robocalls should not be delivered.
- Malware should not be delivered.
- Two endpoints can willingly participate in illegal communication. This should be prevented, or in some cases recorded for further investigation or evidence in legal proceedings (the industry term for this is "lawful intercept").
- Two endpoints can willingly participate in communication that violates parental controls, which should be prevented.
- Operators of enterprise networks know which employees are using which machines for which purposes. Often they configure their networks to prevent unnecessary communication, which is probably a mistake and may be an attack. For example, machines used by engineers should not have access to the enterprise's personnel database.

- Port scanning is the process of trying every TCP or UDP destination port on an IP endpoint, to see if it will accept a session initiation. Port scanning does not in itself do much harm, but should be prevented because it is gathering information to be used in launching other security attacks. (Most malware targets a known vulnerability in a specific program or application.)

Malware is particularly dangerous when it attacks the control mechanisms of a network, such as its directory and routing protocol, or basic utilities such as clock synchronization and certificates. These can be straightforward DoS attacks, or something more insidious. Attacks on directories and routing can be parts of other attacks, because they are such fundamental building blocks that other defenses rely on them. By subverting directory entries or forwarding tables, attackers can draw packets with other destinations to themselves. Having done this, the attackers can read, absorb, inject, or alter packets as they are transmitted (these are the threats to transmission enumerated at the beginning of §1). Attackers can also impersonate the intended destination, thus stealing commerce or secrets.

2.1.3 Protecting freedom and privacy

Endpoint security keeps the content of network communications secret, but it does not hide the fact that the endpoints communicated. Even with encryption, observables such as packet headers, packet size, and packet timing yield plenty of information. These packet attributes are observable by the network infrastructure as a matter of course, and may also be observable by third parties who tap a wire, put a wireless receiver near a wireless transmitter, connect to a wired broadcast medium such as as an Ethernet or cable network, or penetrate an infrastructure machine. "Packet sniffing" software is readily available to help them do it.

Network infrastructure can use the observable information to monitor and censor the network activities of users. Endpoints such as Web servers can use the information to keep track of who is accessing the servers. Third-party snoopers can use it for personal or commercial surveillance.

All of these uses can compromise the freedom and privacy that network users have a right to, from a legitimate or ethical viewpoint. What some national governments consider law enforcement, others consider retaliation against political dissidents. So there is value in building technology to help users evade censorship and protect their privacy. But it is very important to note that these are social and legal, rather than technical, distinctions. The constraints new technology is seeking to evade may be exactly the same constraints that technology was seeking to enforce in §2.1.2. The best technology, in keeping with the "tussle" philosophy of [9], is technology that accommodates all possible outcomes of social, legal, and commercial debates.

Personal data privacy is a related issue that is much more widely discussed. People today are concerned about the massive amounts of personal data that

is collected about them by Web sites and other applications. This data is extremely valuable for selling advertising, and can also be used for worse purposes. Network privacy—privacy about one's usage of a network—can contribute to personal data privacy, but only in a limited way. For example, people can access search engines and read Web sites anonymously, at the cost of longer delays and worse search results (because they are not customized). On the other hand, people cannot participate in social media or electronic commerce in any meaningful way while preserving anonymity.

2.2 Security by packet filtering

Packet filtering is by far the most important mechanism for infrastructure security, and can be used to achieve goals of all types. For packet filtering, the network infrastructure ensures that some or all packets are forwarded through infrastructure members that perform filtering (they may also perform other functions such as forwarding). A *filter* looks for packets that satisfy its *filtering criteria.* On finding such packets, the filter takes some action as an exception to or as an addition to merely forwarding them.

In this section we look at how packet filtering works in a single network—or in a layer of homogenous bridged networks such as the Internet. In §2.4 we will return to the compositional view, considering how packet filtering interacts with other network mechanisms, and where it should be placed in a compositional network architecture.

The first subsection of this section gives an overview of basic packet filtering. Subsequent subsections explore major issues in more depth.

2.2.1 Filtering basics

The oldest filters are *firewalls*, dedicated so-called "appliances" positioned at or near the edges of a network. Their filtering criteria are predicates formed by combining mainly atomic predicates on the values of IP and IP-session-protocol header fields. Their function is to drop disallowed packets. For example, suppose that a firewall is intended to allow only outgoing Web accesses, which of course require outgoing DNS queries. The direction of a packet (inbound or outbound) can be determined from its source and destination addresses or from the link on which it arrives. The firewall might be configured with these four rules:

1. Drop all outbound TCP packets unless they have destination port 80.
2. Drop all inbound TCP packets unless they have source port 80 and the TCP ACK bit is set.
3. Drop all outbound UDP packets unless they have destination port 53.
4. Drop all inbound UDP packets unless they have source port 53.

In the second rule, the ACK bit indicates that this packet is an acknowledgment of a previous packet, meaning that it is not a TCP SYN packet.

These rules are sufficient for the purpose if all packets through the firewall obey the TCP protocol exactly, but of course an attacker may not be so polite. A safer approach would be to make the firewall stateful by having it maintain a table of all ongoing TCP connections. Then the second rule above would be replaced by "Drop all inbound TCP packets unless their source and destination addresses and ports identify them as belong to an ongoing TCP session." Stateful firewalls are often combined with NAT boxes, because NAT boxes sit at the edges of networks and already maintain tables of ongoing sessions.

This brief description contains or implies answers to the basic questions one should ask about a filtering mechanism, namely:

- What are the filtering criteria? For firewalls, configurable predicates on IP and IP-session-protocol header fields. If the firewall is stateful, the predicates can refer to a table of ongoing sessions.

- What actions are taken by the filters? Firewalls either drop or forward ("accept") packets. In addition to these actions, a filter can record packets, raise an alarm, or divert the packets for further analysis. If there is uncertainty about the packets, a filter can rate-limit them or downgrade their forwarding priority rather than dropping them. Rather than dropping session-initiation requests, a filter could reply to them with refusals, which would discourage retries. A refusal to a TCP SYN (request) is a TCP RST (reset). A refusal to an HTTP request is an error code.

- Which packets are filtered, which filter do they go to, and how are they steered into their filter? Firewalls are located at network edges, where every packet going into or out of the network necessarily passes through them. If a firewall is stateful, it is crucial that all packets of a session pass through the same firewall. This property is called "session affinity."

- How are the filters themselves, as potential traffic bottlenecks, protected from DoS attacks? Often firewalls are large machines, with capacity sufficient to handle all their network's traffic, even during a flooding attack. Because these firewalls are dedicated machines, without many programs or control interfaces, they cannot easily be penetrated by malware.

Stateless firewall functionality is sometimes implemented in forwarders rather than separate appliances, in which case the rules are called "access control lists."

For more sophisticated filtering (see §2.2.2), networks often use commercial products known as "intrusion detection systems" and "intrusion prevention systems." The difference is that detection systems only raise alarms, while prevention systems automatically take action against suspected attacks, such as dropping or rate-limiting packets. It might seem that automatic action is always better (it is certainly faster), but there are good reasons for keeping operators and network customers in the decision loop. If a suspected attack is a false positive, or even if its source is uncertain, much legitimate traffic may be dropped. If an operator deploys additional resources on behalf of an enterprise

customer that is under attack, the customer will have to pay for them. The defense against a suspected attack may even be a counter-attack, which is wrong and even dangerous (in a military setting) if not well-justified.

2.2.2 Filtering criteria

Filtering criteria choose the packets on which a filter takes action. The problem of finding filtering criteria is somewhat different for the two major purposes of filtering.

If the purpose of filtering is to prevent or spy on specific communications, then the filtering criteria must describe the specific communications. However, many of these communications are not as specific as we would like. "Signature-based" filters such as spam filters, virus scanners, and parental filters look for keywords, sometimes keywords in specific positions, and other known attack patterns. They can also be stateful, and check whether protocols are being followed. These filters can be valuable commercial products because of the intellectual property in their filtering criteria. Like all security software, to be effective, they must be kept up-to-date. Even so, they cannot detect new attacks, and may be fooled by minor variations on old attacks. For one example, the attacker might fragment packets to hide their resemblance to the signature. For another example, a keyword in email text can be spelled creatively.

One advantage enjoyed by filters for preventing specific IP communications is that attackers cannot usually hide by spoofing. If the attack requires communication in both directions, then the attacker's source name must usually be correct. (In more sophisticated attacks, the attacker can give a false source name and still receive return packets through route hijacking (§2.3.3) or packet sniffing.) Emails (in email application networks) are always one-way, however, so source names can be false.

If the purpose of IP filtering is to prevent flooding DoS attacks, it must contend with the fact that packet source names can be false. On the other hand, at least having a false source name is a straightforward packet-filtering criterion, if the filter can detect it. "Ingress filters" in IP networks check incoming packets to see if the prefixes of their source names match expectations. "Unicast reverse path forwarding (URPF)" in a forwarder accepts a packet's source name as valid only if its forwarding table specifies forwarding *to* the source name on the same two-way link on which the packet arrived. Unfortunately URPF cannot be used in the core (high-speed backbone) of a large network, because routes there are not necessarily symmetric. The Accountable Internet (see sidebar) is a network design with the principal goal of filtering out spoofed packets.

With the possible exception of their source names, the packets of a flooding DoS attack will look benign, so filtering criteria are computed by statistical algorithms. These algorithms look for anomalies, *i.e.,* variations from normal patterns of bandwidth use, protocol use, and other traffic attributes. Needless to say, there is a great danger of detecting too many anomalies ("false positives"), in which case many legitimate packets may be filtered out. Also needless to say, there are widespread hopes that machine learning will improve the precision of

anomaly detection.

In general, the quality of filtering criteria is a limiting factor in the use of filtering to handle IP flooding attacks. The mechanisms for hiding attackers (§2.1.1) are effective. Almost all filtering works on the outermost IP header, regarding the rest of the packet as payload, so layering and encryption in the payload can conceal the true nature of the traffic. Because of these limitations, much of the research on DoS attacks aims to make filtering criteria precise by recognizing certain packets as desirable and rejecting all other packets. We'll call this approach "positive filtering" because the default action on a packet is to drop it, and matching a filtering criterion allows the packet to be delivered. In addition to precision, positive filtering has the advantage of preventing flooding DoS attacks, rather than reacting to them well after they have begun. Positive filtering is complex enough to deserve a section of its own.

2.2.3 Positive filtering

At least one kind of positive filtering is familiar and normal. Some email filters drop emails unless their source name is already a known correspondent of the destination.

A relatively simple kind of positive filtering is performed in a private IP network with software-defined control, as in Ethane [7]. An Ethane controller is a central network member with very complete knowledge of its network, especially the user members. For each user member the controller knows the IP and MAC addresses, the forwarder port to which the member is directly attached, and the user of its machine. It also has policies governing which user members can reach which user members, the session protocols they can employ, and the middleboxes the sessions must pass through. When an Ethane forwarder receives the first packet of a session, it sends the packet to the controller, which uses its knowledge and policies to choose to either allow or disallow the session. If the session is allowed, the controller installs a tuple for it in the forwarding table of every forwarder in the path of the session.

Other proposals for positive filtering, including TVA [37], apply to mixed public-and-private IP networks. In this section we will concentrate on two proposals that together fit into an interesting and useful pattern.

Concerning Secure Overlay Services (SOS) [23] and Mayday [1], there are two important functional questions:

- What are the criteria for allowing or disallowing sessions?
- Where and how are good packets recognized and bad packets dropped?

With respect to criteria, SOS is intended for use during an emergency situation, when networks are so congested that even benign ordinary traffic must be dropped. The only allowed packets to a given destination come from a few pre-configured sources used by emergency responders. Mayday is a generalization of SOS providing for any approval criteria implemented by a certain class of network members (see below).

The Traffic Validation Architecture (TVA) [37] takes the viewpoint that *every* session should be approved by its acceptor. In other words, at the beginning of every session the initiator requests the right to send packets to the acceptor, and no further packets are delivered until permission is granted. The TVA paper gives some guidelines about how an acceptor might make such a decision. For example, the paper suggests that a public server maintain a blacklist of sources that have recently misbehaved, and accept well-formed session requests from all names not on the blacklist. Acceptance grants a sending capability for a limited time only, so a granted capability that is being abused will soon expire and will not be renewed.

Note that a request to send packets is not much different from any other session-initiation packet, and these requests themselves could be used to launch a DoS attack. TVA handles this problem by rate-limiting request packets to 5% of the total volume. Even so requests from attackers could crowd out good requests. TVA handles this problem by tagging request packets with where they entered the TVA network, putting requests with a specific tag in a specific queue at forwarders, and forwarding fairly from the queues.

With respect to performing the actual filtering, in TVA the work is performed by forwarders. Basically an affirmative response to a send request carries a capability, which is then transmitted in every sent packet so that forwarders along its path will forward it.

The TVA designers were diligent in identifying problems with this mechanism and devising solutions for them. A group of related security problems is solved by having capabilities include a component for each TVA forwarder on the session path, and by having the component for each forwarder be computed using a secret known only to the forwarder. Thus capabilities cannot be forged by attackers, and cannot be transferred to other attackers because they depend on the session path. The problem of weak criteria for accepting sessions is solved by attaching a byte limit and time limit to each capability. This way a wrongly-granted capability does limited damage. A group of related resource problems is solved by introducing various optimizations. There is a state-efficient algorithm for tracking session bytes and elapsed time at a forwarder. To minimize bandwidth overhead, the first capability-bearing packet of a session also includes a nonce. When a forwarder validates its 64-bit component of the capability, it caches the capability along with the nonce, so subsequent packets of the session need carry only the nonce. What happens if the nonce is evicted from the cache or the capability limits are exceeded? In the best practice, the sender anticipates this and requests a renewal before it occurs. Otherwise, packets with no capabilities, expired capabilities, or incorrect capabilities (because the session path has changed) *are* delivered, but with the very lowest priority.

For recognizing good packets and filtering out all others, both SOS and Mayday rely on an overlay IP network of trusted, cooperating members—in a particular deployment instance, these members might not belong to the AA of a particular network, but instead might belong to an enterprise or peer group whose machines cooperate for mutual protection. In addition to the overlay network, a potential target must be surrounded in the Internet underlay by a

ring of ordinary packet filters. These ordinary filters, such as firewalls or filtering forwarders, must have the capacity to handle flooding DoS attacks, and must be configurable by overlay machines or by people representing the overlay. The general idea is that good packets are transmitted to the target through the overlay, and the links of the overlay are implemented by the Internet underlay. The packet filters in the underlay can recognize which packets are also traveling through the overlay, and drop all other packets.

Figure 8 is a graph view of the physical arrangement. All nodes are members of Internet networks, either the protected target's network or other networks bridged to it. Note that all Internet paths to the target go through the filters. Note also that overlay machines can be located anywhere, close to the target or far away from it.

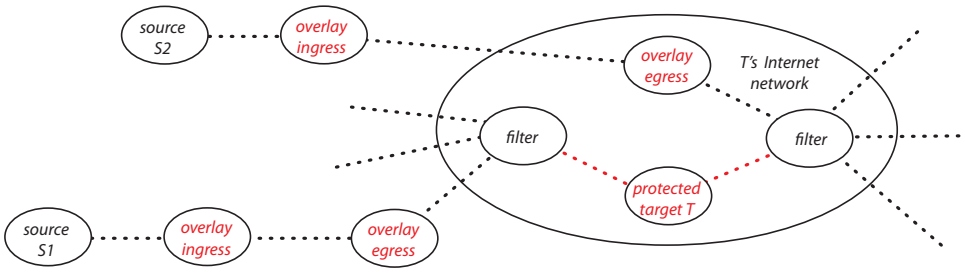

Figure 8: A graph view of Internet members involved in overlay-based positive filtering. The members named in red are on overlay machines, *i.e.,* their machines also have interfaces to the overlay network. The paths in red are the *only Internet paths* to the protected target.

In designing an overlay network for positive filtering, there are three important choices to be made. SOS makes specific choices, while the Mayday paper points out that there are other choices, and evaluates some combinations of them. We now explain the three choices.

Source authentication. This choice concerns how an endpoint authenticates itself to the overlay as a source of legimate packets. In SOS the source is an overlay member, *i.e.,* it has special software. It creates a secure link to an overlay ingress member, using IPsec with endpoint authentication. SOS source members know the Internet names of many ingress members, well-distributed so that they cannot all be overwhelmed by flooding attacks.

Mayday emphasizes an option that is architecturally more complex, but has broader applicability because the source need not be an overlay member (both Figures 8 and 9 depict this option). In this option packets from a source to target name T are routed to some proxy that is an ingress member of the overlay. The proxy accepts the TCP session, and can then authenticate the source by asking for a user name and password associated with the target service. If the source is authentic, the proxy makes a TCP session *through the overlay*

to the target; these two TCP sessions then become two parts of a compound application session.

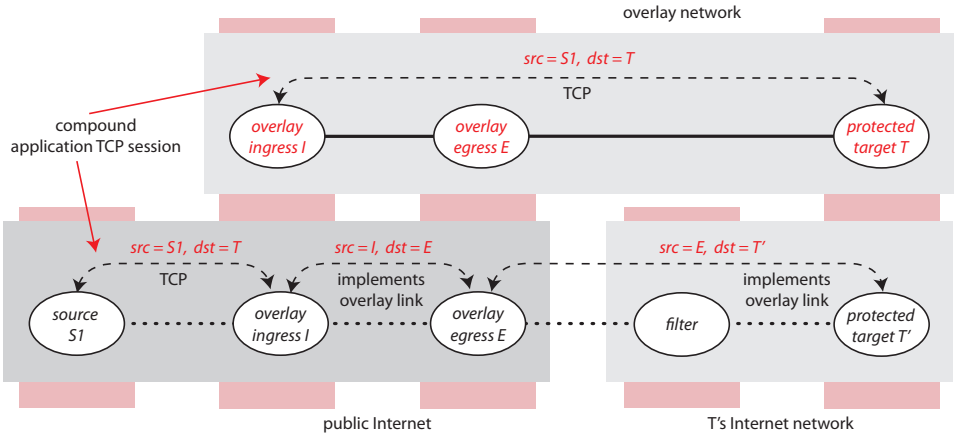

Figure 9: A session view of overlay-based positive filtering, illustrating the following options: source is not an overlay member, target has different names in overlay and underlay, routing is singly-indirect.

Lightweight authenticator. Figure 9 is a session view of allowed access to protected target T. The last overlay hop between an egress member of the overlay and the target is implemented by an underlay path that goes through a filter. The lightweight authenticator is the attribute of underlay packets from egress member to target that causes the packet filter to recognize them as overlay packets and allow them to pass. The simplest lightweight authenticator is the IP name of the egress member (here E) in the source name of a packet; this is what SOS uses. Other authenticators proposed by Mayday include the destination port, destination name, and other header fields whose contents can be manipulated by the egress member.

The critical property of a lightweight authenticator is that it must be a secret—if attackers knew it, they could simply send underlay packets that match it. You might think that the destination name is the worst possible authenticator, but it can be a good one if the underlay name of the protected target is different from its overlay name, as shown in Figure 9, and if it can be changed easily and frequently by local control in the target's network.

Overlay routing. The identities of authenticating ingress nodes are fairly public, as all packets to T must be routed to or destined for them. The purpose of having a full overlay network with its own routing (as opposed to having ingress nodes only) is to vary and hide the paths of packets between ingress members and the target. This keeps attackers from flooding the paths to the target rather than the target itself. It also keeps the identities of egress nodes secret, which is indispensable if the lightweight authenticator is the name of

an egress node. SOS uses egress names as authenticators, and also employs complex routing through the overlay based on distributed hash tables. The paths are long and introduce considerable delay.

Mayday takes the position that effective overlay routing can be much simpler, with options including no routing at all (ingress and egress nodes are the same), and singly-indirect routing (one hop between ingress and egress nodes, as in Figure 9). The Mayday paper reports on analysis showing that certain combinations of overlay routing and lightweight authenticator provide "best cases" for trade-offs among performance and security. For example, designers who want moderate levels of both performance and security should use singly-indirect routing with any authenticator other than source name.

2.2.4 Filtering resources

When filtering is used as a defense against flooding attacks, the filters and the network paths to them must have sufficient resources to handle all of the attack traffic. Thus resources and scalability are important aspects of the design of packet-filtering mechanisms.

In a layer of homogenous bridged networks such as the Internet, we can visualize the graph of all network paths leading from sources to an attack target as a giant tree. Filters should be arranged so that there is a filter (or sequence of them, if filtering is pipelined) on each path. Considering this tree, there are general trade-offs between placing filters closer to the target or farther from the target. The advantages of placing filters closer to the target are:

- The root of the tree is at the target. Obviously, there are fewer paths to cover close to the target, and fewer packets overall for the filters to process.
- Usually the responsibility for protecting the target lies with the AA of the target's network, so the AA has an incentive for deploying resources in its network, close to the target.

The advantages of placing filters farther from the target are:

- If filtering is farther from the target, the damage done by attack traffic is lessened, because attack traffic is carried for shorter distances along fewer links. Note that the damage of a flooding attack is not limited to the intended target, because traffic to many other destinations will also suffer because of congested links.
- If filtering is farther from the target, it is closer to the sources of attack traffic, and may have more information about it. For example, an ingress filter in an IP access network knows the prefix of all genuine source names, so it can filter out packets with false source names. The access network sees all of a suspected source's traffic, so attack patterns are more likely to be detectable. An access network may also know more about the type and reputation of its sources (device type is relevant because some mobile

operating systems and vendor hardware are more easily penetrated than others). More precise filtering means less collateral damage.

- When there is a major flooding attack, it is usually focused on a small number of targets. Very often, the attack packets are coming from a botnet, with a large number of sources well-distributed across the public Internet. So the total amount of available filtering resources near sources greatly exceeds the total amount of resources available near targets.

A third option, filtering in the topological core of the network, is never used because the core is a region of high-speed links and high-speed routers handling large flows of packets. Routers would have to filter at line rate, based on filtering rules stored in expensive Ternary Content-Addressable Memory, and the hardware's capacity to store rules would not be adequate for the number of rules required.

Unfortunately, the many advantages of filtering near sources are balanced by two major disadvantages:

- Networks may not have sufficient incentive to use their resources to protect targets that are remote from them. (However, this is making a big assumption, that topologically distant networks are also distant administratively. If the administrative distance is small, then this disadvantage disappears.)

- Even if source networks are willing to cooperate with target networks, the necessary coordination is not easy. Infrastructure machines such as routers, controllers, and policy servers must cooperate on attack diagnosis as well as filtering. (Before the recent advent of programmable routers, this would not even have been possible.) Control communication between networks must be secure, because attackers could abuse it. Yet ordinary endpoint security may not be applicable, because the networks do not know the identities of forwarders in other networks, nor do they know which forwarder is on which path. Finally, even when communicating correctly with another network, the target network cannot necessarily trust it.

There are proposed solutions to all the problems of incentives and control [4, 10, 28, 36], but they are not simple. Historically, cooperation between networks with different AAs has been scarce [15].

The normal practice is to place packet filters close to the target, because of the incentives. Yet there are proposals for moving filtering activities farther from the target, of which Active Internet Traffic Filtering (AITF) [4] is a representative example.

AITF is concerned with filtering in routers, which must be performed at line rate. The authors argue that the only way to filter fast enough is to store the rules in each router's ternary content-addressable memory, which is expensive and limited in size. Given the problem of attacks by botnets, in which packets are coming from millions of distinct sources (and thus require millions of distinct

filtering rules), there are not enough router resources close to a target to filter successfully.

The AITF solution is to push filtering backward along packet paths to routers close to the sources of attack traffic. If this were done perfectly and completely, then each edge network would only have to filter the sources whose packets the edge network injects into the public Internet, and it would be doing this on behalf of all other public networks.

We will first describe how this is done in its simplest form. Along each path from source to destination, there are filtering routers *sourceGateway* near the source and *destGateway* near the destination. As a packet travels along the path, these IP addresses are added to the packet. If a destination finds that it is under attack, it requests *destGateway* to start filtering out some *(source, sourceGateway, destGateway, destination)* flows. The *destGateway* does this, and also sends requests to the *sourceGateway*s of those flows asking them to do the same filtering for a fixed period of time. If a *sourceGateway* accepts the request, then the *destGateway* can stop filtering the same flow.

There are many ways in which this simple idea must be augmented to make it reliable and secure. First, the request and acknowledgment packets of the AITF protocol itself could be used to flood a network, so they must be rate-limited.

Second, attackers could use spoofing in two ways: (i) an attack node could block legitimate packets from *source* to *destination* by sending filtering requests to *sourceGateway*, pretending to be *destGateway*; (ii) an attacker could flood a destination with packets pretending to be coming through *sourceGateway*, which would hide their true source and might also cause the network of *sourceGateway* to be disconnected from parts of the Internet (see below). To prevent spoofing, both the flow descriptions in packets (naming the gateways) and the packets of the filtering-request handshake carry nonces inserted by the participating gateways.

Third, a *sourceGateway* might agree to filtering and then not comply. The *destGateway* will check that the promised filtering is really happening. In fact additional routers between *sourceGateway* and *destGateway* can put their names into the flow descriptions of packets. If there are other cooperating gateways, and if the *sourceGateway* fails to reply to a filtering request, or rejects the request, or accepts the request and fails to filter, then the *destGateway* can try the next gateway in the flow description. This step is called *escalation*. Escalation can act as a significant incentive, because if *destGateway* asks *secondGateway* to filter in lieu of *sourceGateway*, *destGateway* might tell *secondGateway* to filter out *all* packets from the *sourceGateway*'s network.

The AITF proposal expands on this kind of reasoning to show why networks might have general incentives to help protect targets in other networks. A network under attack may be more willing to accept incoming packets from a cooperating network, because it trusts that the packets have already been filtered. Even if AITF is only partially deployed, the incentives still hold, because networks that cooperate with each other through AITF could still communicate with each other during widespread attacks.

The proposals for moving filtering toward the sources of attack traffic date

from the early 2000s. In the 2010s cloud computing has advanced so far that most filtering is performed in clouds on behalf of target networks. In clouds the filters are virtual machines, and the number of filters expands and contracts with fluctuations in load. It is also practical to employ sequenced filters, directing packets that early filters find suspicious to later filters with more detailed screening.

2.2.5 Proxies

Proxies, and compound sessions formed by these proxies, are commonly used to evade packet filtering. This works because the header fields of the packets in the component simple sessions of a compound session can be completely different. Although joinboxes can also form compound sessions, they are not endpoints of session protocols, and therefore cannot do all the manipulations we will see proxies doing in the remainder of this session.

Perhaps the oldest example of such a proxy is an "application gateway," which is installed in a private IP network for the benign purpose of evading the too-simple filtering imposed by the firewall. For example, an enterprise firewall may block all outgoing sessions except Web accesses. However, the enterprise may also wish to allow outgoing sessions of another kind, when they are initiated by specific users. The firewall cannot enforce this policy because it does not know the mapping between internal IP names and users (and the mapping may not even be static).

An application gateway for the application, for instance Telnet, solves this problem. To use it, a user initiates a Telnet session to the application gateway inside the enterprise network. By means of an extension to the Telnet protocol, the user supplies a password to authenticate himself to the proxy, and also the name of the real Telnet destination. The proxy initiates a Telnet session to the real destination outside the enterprise network, and joins the two simple sessions in a compound session. The enterprise firewall allows outgoing Telnet sessions from the application gateway only.

Proxies are the principal tool for providing users with privacy, anonymity, and the ability to evade censorship. We will show how this works in three stages.

The first stage is simplest. If, for example, a user wishes to access a Web server with some privacy and anonymity, the user's browser requests from the IP interface on the user machine a "proxied TLS" session with a friendly proxy outside his home network (Figure 10). A "proxied TLS" session is just like a normal TLS session except that: (i) instead of looking up the domain name *dangerous.com* and using its IP address as the destination of the session, the IP interface uses the proxy's address as the destination of the session; (ii) it expects and verifies the certificate of the proxy, not the Web site.

After the proxied TLS session is set up, it appears perfectly normal to the user's machine. The proxy, however, decrypts the HTTP request, and uses it to make another TLS session with the actual server. After this the proxy relays packets between the two components of the compound TLS session. Because of the compound session, the IP network of the user does not know what the user

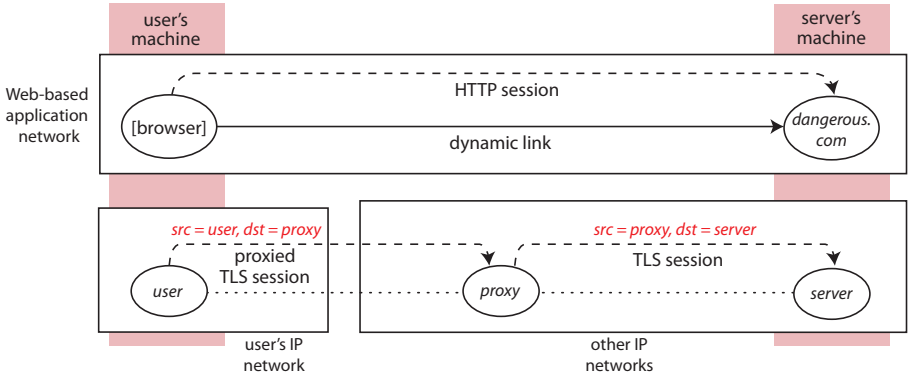

Figure 10: A proxied TLS session protects the user's privacy in his home network, and anonymity at the Web server.

is connected with, and the final destination does not know who the user is.

One remaining disadvantage of the first stage is that the user has no privacy from the proxy. The other disadvantages come from the fact that the addresses of helpful proxies must be publicly available (so users can find them) which means that they are available to the user's adversaries as well. Most importantly, if the user's IP network is censoring the network activity of its users, it can simply block packets addressed to external public proxies.

The second stage of proxy use is intended to evade censorship in a network, such as a national ISP, that seeks to limit or spy on the network activity of its users. There are several similar proposals [18, 19, 35], all using proxies, but in a way that still works even if the censoring network is blocking access to known proxies. As a representative of these proposals, we will present Cirripede [18].

As shown in Figure 11, the ISP of the Cirripede client is filtering out packets from the client to certain Web servers, represented here by the "covert destination." The client cannot evade this censorship by using a false source name, because then replies from the Web server will not be delivered to the client (also, the network may be blocking *everyone's* access to the site).

For Cirripede to work, there must be a network operated by a friendly AA on the path between the client's ISP and many destinations, including the covert destination. The client must first signal to the friendly network that it wishes to use Cirripede for a time interval. To register as a Cirripede client, it must use a "covert channel" of communication, one that its ISP is unlikely to recognize. Covert channels are constructed from attributes of a packet stream that the sender can control and observers of the packet stream are unlikely to notice, including packet timing, order of TCP packets, unused packet fields, and pseudo-random packet fields. (Covert channels are always low-bandwidth,

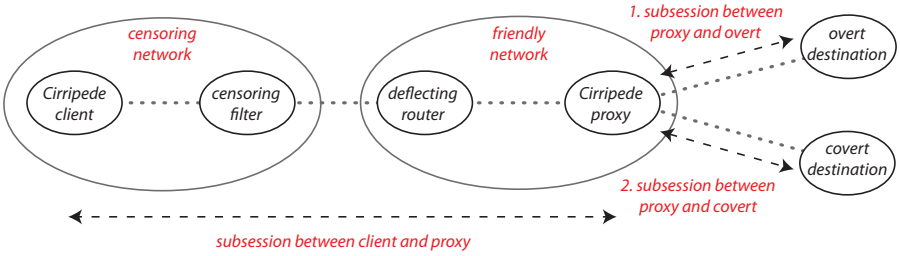

Figure 11: Subsessions of a session between a Cirripede client and a covert destination. In the subsession on the left, names in the IP header are those of the client and overt destination; packets from the client are deflected to the proxy as an exception to normal forwarding.

and vulnerable to detection if popular, which is why they must be used sparingly.) To signal covertly that it wishes to register with the friendly network, the Cirripede client puts a secret number in the pseudo-random initial-sequence-number field of a TCP SYN packet. The packet is intercepted by a router in the friendly network on the path from the client, and converted to a registration for the client. As a result of the registration, a special "deflecting table" in friendly routers over-rides normal routing. When subsequent packets from the client reach these routers, they are sent to a Cirripede proxy.

The Cirripede proxy will help the client access the covert destination, but it takes a complex session protocol to do this safely. First, the client requests a TLS session with an overt destination—one that the client is allowed to reach—and completes a TLS handshake with it. During this phase the proxy is present in the path but takes no active part. Even if the censoring filter is monitoring every packet of the TLS handshake, it will see nothing out of the ordinary. After the TLS handshake all packets to and from the client will be encrypted, so their content need not be constrained. During the entire TLS session, the packets seen by the filter will have the overt destination in their source or destination field.

Next, the proxy will end the session between itself and the overt destination, and inform the client that the proxy is in the path of the session. The client will respond by sending it the name of the covert destination it actually wishes to reach. The proxy initiates a TLS session to the covert destination, and afterwards relays packets between the client and covert destination.

Just as the censoring authorities could learn about friendly proxies and block their IP addresses, could the authorities learn about the helpfulness of the friendly network and refuse to bridge with it? The assumption behind Cirripede is that the friendly network is on the censoring network's path to large parts of the Internet. The prediction is that the censoring network will not cut it off, because the effects would be too public and too draconian.

The remaining vulnerabilities are that the user still has no privacy from

the proxy, and that that Web site or other destination knows that the user is connecting to it through a proxy (again, because proxies are publicly known). The latter issue will be discussed in §2.4.3. Privacy from the proxies can be obtained by using the Tor service.

Tor [11] is a well-known public service for providing users with private, anonymous sessions to Web sites and other services. Volunteers worldwide lend their machines' resources to act as Tor proxies. Tor proxies form an overlay network with its own randomized routing (as in §2.2.3) to carry the users' traffic. Most importantly, the initiator of a Tor session chooses proxies for the session, and each Tor proxy has a public key. Packets of the session are encrypted as many times as there are proxies, working from the key of the last proxy to the key of the first. Each proxy decrypts one layer of encryption, so that it knows the names of adjacent proxies in the chain, but nothing else.

2.3 Security that is not packet filtering

The majority of infrastructure security is implemented with packet filtering, which is a general mechanism that can be used for many purposes. In this section we review a few techniques for infrastructure security that are *not* packet filtering.

2.3.1 Resource replication

Current network technology, has made it both feasible and popular to defend services against resource attacks by replication of resources, so that there are enough resources to withstand attacks. This is easiest in a cloud, where a service under attack can quickly be granted more virtual machines.

It is even better if resource replicas are geographically distributed, so that some replicas can be reached when other parts of the network are too congested. Because attacks on DNS servers are so common and damaging, it is especially important to have distributed authoritative DNS servers for popular domain names. Queries are distributed across the replicas by means of DNS anycast or IP anycast. If there are five replicas sharing the load and one has been overwhelmed by an attack, DNS or IP anycast may not be dynamic enough to redirect queries away from the failed replica, but at least queries directed by anycast to the other four will succeed.

Resource replication works best when replicated data is fairly static, and it is not feasible when queries update the data. (Note that this remark is specific to the context of replicating data as a defense against flooding attacks in the public Internet. In more private distributed systems, data is frequently replicated for reliability, and there are many protocols for making data that has been updated at a single replica consistent across all replicas.) In cases where data is dynamic or updated by queries, it can be distributed across multiple sites by sharding, *e.g.,* by partitioning the keys of a key-value store across sites. This will increase the total resources available, and also the expected availability of an arbitrary key.

2.3.2 Reduction of amplification

Another defense against resource attacks is to reduce the amplification factor on which they depend. For example, It has become common for servers to defend themselves against SYN floods (§2.1.1) with "SYN cookies." In this defense, the server returns a SYN+ACK packet with a specially-coded initial sequence number (the cookie). It then discards the SYN, using no additional resources for it. If the SYN was an attack, it has not been amplified. If the SYN was legitimate, on the other hand, it will elicit an ACK from the initiator with the same initial sequence number incremented by one. By decoding the sequence number, the server can reconstruct the original SYN and then set up a real TCP connection.

A flood of DNS queries is amplified when servers query other servers. A very effective defense against these attacks is longer times-to-live for cache entries, perhaps 30 minutes, in recursive and local DNS servers [29]. If local entries are cached longer, there will be fewer queries and retries made to authoritative servers. There are many good reasons for DNS cache entries with short times-to-live, but these can be changed as an adaptive measure during attacks. The same idea would work for many other services with caching.

2.3.3 Endpoint security for control protocols

Control protocols are used to maintain and distribute network state. It should be no surprise that unauthorized update or distribution of network state can be used for malicious purposes. The most well-known attack of this kind is "BGP hijacking," in which an attacker advertises an IP prefix it does not own as a way of drawing packets for that prefix toward itself.

The defense for a control protocol is endpoint security. For instance, Border Gateway Protocol Security is a security extension to BGP that provides cryptographic verification of messages advertising routes. Similarly, Domain Name System Security Extensions protect DNS lookups by returning records with digital signatures.

The use of TLS or ESP is not always possible for control protocols. An endpoint may not have a certificate or other credential to prove its identity. The protocol might require high-speed, high-volume operation. Or, the protocol might simply be too old to incorporate endpoint security, even if it is feasible.

In these cases there are lighter-weight measures that can help. Network members that make requests should keep track of their pending requests and not accept unsolicited replies. Replies should be checked for credibility, whenever that is possible. Most effectively, a network member can include a nonce or random field value in a session-setup message or request. Subsequent messages of the protocol must have the same nonce or random value, so that no attacker without access to the previous messages of the session can send messages purporting to be part of the session. Without the nonce, an attacker could do something to trigger a query, then send a spurious answer to the query.

2.3.4 Very special-purpose techniques

In special cases, undesirable communication can be prevented by making a user member unreachable, by altering either the network's directory or the network's forwarding. For example, an illegal gambling site can be made unreachable in the public Internet—at least to those who do not know its IP address—by removing or falsifying its DNS entry. This is a special case because, although the gambling site is connected to the public Internet, officials want to prevent *all* communication with it.

2.4 Compositional infrastructure security

This section is concerned with the interactions among infrastructure security in single networks, composition operators, and other aspects of networking.

2.4.1 Bridging

When networks are bridged, they have either *shared links* or *shared members*. If a member is shared it belongs to both networks, and may play a different role in each. For instance, a gateway to a private network may be shared between the private network and its ISP's network; in the private network it is an infrastructure member, while in the ISP's network it is a user.

For infrastructure security, the most prominent threats occur among the many bridged networks of the public Internet. There is much less concern about the security of Ethernet LANs, particularly wired Ethernets, because they are usually isolated—their member machines usually connect to the outside world through IP networks at a higher level. Wired Ethernets can be attacked, but the attackers must have some kind of physical access to the machines connected to them [25]. In addition to attacks by insiders, a visitor to an enterprise might plug into an unused wall socket or switch a cable from one machine to another. A determined attacker might even even drop a memory stick with a company's logo in the company's parking lot; someone who finds it might plug it into a company computer to see who it belongs to.

2.4.2 Layering

A machine could be a member of several independent networks and, because of fate sharing, be attacked through any of them. Most often, however, each packet arriving at a machine is being transmitted through multiple layered networks simultaneously, for example an Ethernet LAN, an IP network, and an application network. Amplification of a flooding attack, or damaging processing of a malware packet, can take place at any of these levels. Filtering can also take place at any of these levels.

First we consider the networks layered below the filtering network and implementing its virtual links. If the filtering is to achieve what we assume it is achieving, then for all links on paths between a filter and a protected target, it must be true that no packet is received on the link that was not sent on

the link. In other words, lower-level networks do not inject packets into the implementation of the link.

This might seem like a fanciful concern, but it is very real in clouds. The filtering network might be a tenant of a multi-tenant cloud, in which case its links are virtual channels implemented by sessions in a lower-level cloud network that is shared among tenants. If the cloud network does not isolate tenants properly, then cloud-network members of another tenant might pretend to belong to the filtering network. They might insert packets into filtering-network sessions in the cloud network, or manipulate packets in them before they are delivered to the session endpoint. Even if the other tenant is an honest enterprise, its virtual machines may have been penetrated by attackers (which is easy to do, see the sidebar on botnets). Even though the AA of the filtering network is associated with the tenant, the integrity of its links is the responsibility of the cloud network, whose AA is the cloud provider. In another scenario, a penetrated Ethernet can also inject packets into the links of networks layered on it [25].

It is common to deploy IP-based intrusion detection systems that look into the payloads of IP packets for attack signatures at higher levels, for example signs of malware at particular locations in the payloads. These systems have the limitation that they are assuming a known and fixed layer architecture between themselves and the endpoints they are protecting.

In many ways it makes more sense to include filtering middleboxes in session paths in each network separately, thus making no assumptions about which other networks a packet is traveling through. A network-specific filter, particularly for an application network, can rely on far more knowledge of the application and network members. Today virtualization technology makes this approach far more practical than it was in the past. Of course, higher-level filtering can augment lower-level filtering, but cannot replace it.

The overlay architectures in §2.2.3 offer another attractive possibility. If an endpoint machine should receive *only* packets transmitted through a higher-level (overlay) network, then the overlay can filter, and the implementation underlay network can be configured to allow only those underlay packets that are also overlay packets.

2.4.3 Middleboxes

An important interaction between endpoint security and infrastructure security has already been covered. Packet filters are middleboxes, so all the observations about encryption and middleboxes (§1) are relevant.

In general, a middlebox can alter a packet stream that passes through it. For this reason, only trusted middleboxes should be placed in packet paths between a packet filter and a target it is intended to protect.

The interaction between filtering and proxies is of paramount importance. With proxies, a session can consist of a chain of multiple simple sessions, each of which has completely different headers. As we have seen, if the purpose of filtering is to prevent flooding attacks, then there is some flexibility as to where

the filtering can be located. So effective filtering must be located where an attacker's sessions can be identified by the headers of the simple sessions in that locality.

If the purpose of filtering is to block or spy on specific communications, then it can be evaded by making the simple session of a compound session look innocuous in the blocking AA's network. This statement covers not just evasion of censorship in a user's access network, but also supposed spying by the proxies themselves (one of the threats addressed by Tor). The AA of a user's access network can exercise some control over this by choosing not to bridge with networks harboring proxies, but this mechanism of control is limited by the AA's need to maintain connectivity with all or most of the global Internet, with reasonable performance and cost.

Unfortunately the Tor mechanism for privacy has one deficiency with no current solution. At the far end of the session, the acceptor of the session can know that Tor is being used, because there are readily accessible and regularly updated lists of Tor nodes (necessary so that prospective users can find them). Fraudsters, spammers, and other criminals are big users of Tor, along with law-abiding people in need of privacy. Consequently an increasing number of services are rejecting or otherwise discriminating against Tor users [24].

2.4.4 Routing

Wide-area routing frequently creates different paths for packets traveling in different directions between the same two endpoints. Even packets traveling in the same direction may be spread across multiple paths because there has been a failure in one of the paths, or a need for better load-balancing.

Thus routing requirements can conflict with the need for session affinity (§2.2.1), *i.e.,* with the need to have all packets of a session pass through the same filter. These needs are easiest to reconcile within a cloud, where paths come together and there can be a mechanism for sending all packets of a session to the same virtual instance of a filter.

2.4.5 Session protocols

SYN cookies (§2.3.2) are very clever, but like most clever solutions, they come with hidden limitations. A server using SYN cookies effectively drops all optional information in TCP SYN packets, which means that any network capability or feature relying on extensions to TCP is disabled. Note that many servers using SYN cookies are Web servers, and many new features relying on extensions to TCP (such as Multipath TCP, just to name one example) have improved Web access as a major use case. These issues are also discussed in the chapter on *Session Protocols*.

There is another defense against SYN floods that is less efficient than SYN cookies, but comes with fewer limitations. This defense uses a proxy in the path to a Web server that stores and responds to SYN packets, but does nothing else with a SYN packet until it receives the ACK that completes the handshake. On

receiving the ACK, the proxy forms a new session by sending the SYN to the server, and subsequently acts as a transparent proxy between the two sessions. If the proxy does not receive a timely ACK, then the SYN packet was part of an attack or the client has failed, so the proxy drops it.

3 The verification challenge

There is growing interest in verifying the correctness of networks, most importantly with respect to security. For example, there has been significant progress on verifying the forwarding tables of an IP network—or the algorithm that computes them—to ensure that they satisfy reachability and access-control policies [3, 5, 20, 21, 27].

It should be abundantly clear from this chapter, particularly from the sections covering the interactions among composition operators, security mechanisms, and diverse network mechanisms for requirements other than security, that this work is necessary but not sufficient.

If this is not convincing enough, one need only look at the hundreds of new papers on security published every year. Each responds to a specific known or suspected security threat with a specific defense—which is often complex. It is hard to believe that network operators will ever be able to deploy all of these defenses, and troubling to think that they will have to choose some arbitrary subset of them. Furthermore, many of these security papers are excellent. They use careful and subtle reasoning to enumerate possible attacks and discuss which ones their defenses should hold against. But there are so many that one is driven to say, "What an insightful list! But how do I know that they thought of *everything*?"

We feel that there is an urgent need to start integrating and unifying this knowledge, and that compositional architecture provides a framework for doing so. We are beginning by analyzing results from a large number of research sources, and unifying them through use of the model. Further steps will leverage the structure and modularity of the model to reduce the "attack surface" of networks that satisfy today's requirements, so that they can have property guarantees built into their software and hardware.

Exercises

1. In the example VPN, what forwarding rules are needed to forward packets of the session in both directions? Assume that the virtual link between sites uses a dedicated "VPN service." At each member, a link can be named unambiguously by its direction and other endpoint; use "external" as a shorthand for some external outgoing link. Write each rule in the most general way, using wildcards.

2. In [8], Clark suggests the principle that, in layered architectures, timeouts at a higher level should always be longer than timeouts at a lower level. (The idea is that lower levels can repair faults quickly and locally, for example by routing

around a failed local link. If they fail, then higher levels can apply more global or more radical recovery techniques.) Where in this chapter is Clark's principle violated, and what is the result?

3. This exercise concerns overlay-based positive filtering of TCP traffic. The lightweight authenticator is the combination of destination TCP port and destination name, where destinations are found in a public /24 subnetwork of the Internet. There are several ways that an attacker can probe a subnetwork, finding out whether or not a SYN packet reaches a server [1]. Let's say that a probe requires 7 packets on the average, probe packets are 130 bytes on the average, and the subnetwork is served by a 100 Mbps Ethernet. What is the minimum time it would take for an attacker to check all possible lightweight authenticators, to find the one that gets through the filters?

4. Both TVA (§2.2.3) and AITF (§2.2.4) propose that packet filtering against flooding attacks be performed by IP routers. Why does AITF conclude that filtering rules must be stored in fast ternary content-addressable memory, while TVA does not?

5. Explain in detail how VPN technology fits the overlay pattern for positive filtering. Are there differences from SOS and Mayday as examples of the pattern?

6. Read the SOS paper [23] and assign a unique name to each object or object type referred to in the SOS architecture. For each of these object or object types, what is the set of names used for it in the paper? What do you think of the technical writing in this paper?

Appendix A: Network members

A *member* of a network is a software or hardware module running on a *machine.* The member participates in the network and implements some subset of its protocols. The member can also be thought of as the machine's interface to that network.

A network is assumed to have a single *administrative authority* that controls it and is responsible for its services. Distinguishing between two networks because they have different administrative authorities is making a social or commercial distinction, and is not technical in any way. It can also be a rough approximation, for example in the case of a peer-to-peer network, whose members are a loose association sharing rules of behavior but not trust. Nevertheless, "administrative authority" is a simple way to begin talking about issues of authority and responsibility.

This assumption partitions the members of most networks into two categories: *infrastructure members* that are controlled by the network's administrative authority, and *user members* that are not.

A *forwarder* is a network member whose primary task is forwarding, although it may do other tasks as well. Usually a forwarder is an infrastructure

member. In different networks forwarders are often given different names, such as "switch" and "router," even though they play exactly the same role. The task of forwarding does not entail any alterations to the packets being forwarded; if there are such alterations, then the forwarding task is being bundled with other tasks in the same network member.

Appendix B: Session-protocol composition

Many communication services are provided by, or with the help of, *session protocols*. A session protocol for point-to-point sessions is a set of rules for the format and sequence of packets exchanged between the two endpoints of the session. The design of a network includes one or more session protocols.

A *message* is a semantic entity within a session protocol. Because of the conversational nature of protocols, it would be unusual for a packet to contain multiple messages, but length restrictions could easily cause a message to be transmitted in multiple packets. *Control messages* are used by a protocol to synchronize the endpoints and share specific parameters. *Data messages* contain the substance being communicated. Although a session protocol can have only one of these message types, many protocols have both, or mix control information and data in a single packet.

Within a network, session protocols can be composed, so that the same session benefits from the services implemented by multiple protocols. When two session protocols P and Q are composed, one of them is *embedded in* the other. If P is embedded in Q, for instance, most packets in the session will have the format shown in Figure 12, in which the P header and payload are encapsulated in a Q payload (the figure also shows optional footers, which are required by some protocols). In addition, the session may include control messages of Q that are independent of P and have no encapsulated P messages.

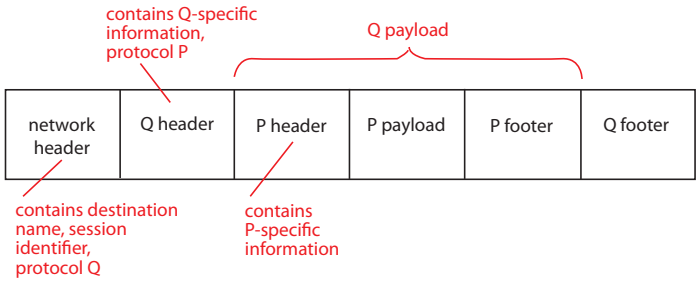

Figure 12: Packets of a network with session protocol P embedded in session protocol Q. Protocol footers are optional.

As in the figure, the first header of a packet has a format dictated by the network design, and includes the destination name and session identifier for the

entire session, so that all packets of the session will be identifiable as such to the forwarders. Each network or protocol header names the type of the next header, if any, so that network members can parse and handle the packet with the appropriate protocol stack.

Figure 13 shows the headers of a packet observed in the AT&T backbone, with annotations showing the layered networks through which it is being transmitted [38]. We can see that the General Packet Radio Service (cellular) network design is based on the IP protocol suite, as is the public Internet (of course) and the VPN. Cellular packets begin with IP headers, followed by the headers of two session protocols, GTP embedded in UDP.

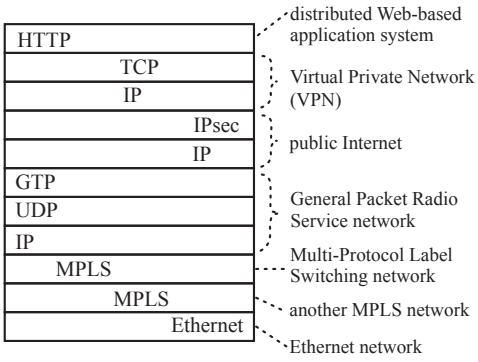

Figure 13: Headers of a packet observed in the AT&T backbone.

Appendix C: Compound sessions

The classic Internet architecture does not recognize *middleboxes*, which are network members—other than forwarders—in the paths of sessions. Middleboxes are ubiquitous and essential in today's Internet, performing a wide range of useful functions for security, interoperation, and performance optimization. In many networks, the number of middleboxes is comparable to the number of forwarders.

The most common means for inserting middleboxes in the path of a session is routing and forwarding. Forwarders forward each packet of a session first through a sequence of middleboxes, after which it is forwarded to its destination. If the session is two-way, the paths in each direction need not be the same, and the middleboxes in the paths might differ as well. In the remainder of this section, however, we will consider only two-way sessions in which each middlebox is in the session path in both directions.

Figure 14 shows an interesting middlebox that is always inserted by forwarding: a Network Address Translation (NAT) box. The figure shows that an IP member with private name V has initiated a TCP session to public name *P2*.

P2 cannot send packets directly to *V*, because a private name is ambiguous in the public Internet. The solution to this problem is a NAT box in *V*'s private network with its own public name *P1*. As packets travel from from left to right in the figure, the NAT box changes their source name from *V* to *P1* and changes the session identifier (source port in TCP) to a value unique within the NAT box. The NAT box maintains a unique association between the two subsessions, so as TCP packets travel from right to left it is able to change their headers from that of the rightmost subsession to that of the leftmost.

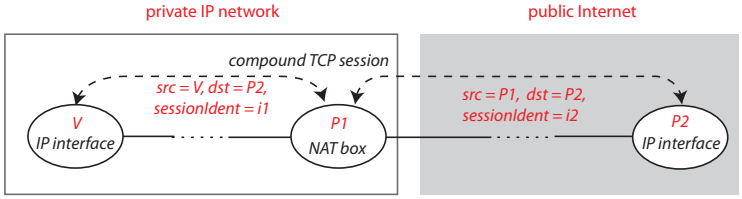

Figure 14: A compound TCP session formed by a NAT box, which is acting as a joinbox. *V* is in the private IP name space, while *P1* and *P2* are in the public IP name space.

Both endpoints of this session observe it as a single session, yet because of the NAT box there are two different source names in the forward direction, and two different session identifiers. Because of these significant properties, we call the NAT box a *joinbox* and the session a *compound session* composed of two *simple sessions*. In a simple session, all packets in the same direction have the same source, destination, and session identifier.

Note that even though the NAT box is inserted in the path of forward packets by forwarding, it is not inserted in the path of reverse packets by forwarding. Rather, the reverse packets name it as their destination. This is the other means for inserting a middlebox in the path of a session.

In addition to being composed by a joinbox, two adjacent simple sessions can also be composed by a *proxy*. A proxy is more powerful than a joinbox, because it is a protocol endpoint. This means that a proxy acts as an endpoint for two "back to back" sessions—it is the acceptor of one session and the initiator of the other. For interoperation, a proxy could even translate between two completely different session protocols! A more common example of a proxy is a Web cache (assuming HTTP, not HTTPS). The cache first accepts a TCP session from a client, and receives enough packets to read a full HTTP request. If the cache can supply the requested page, it simply does so. If not, the cache initiates another TCP session to the proper server. The two TCP sessions are associated at the proxy, so that all buffered and future packets from the client are sent to the server, and all packets from the server are sent to the client. At the same time, the cache saves the fetched Web page for future use.

Who controls the formation of a compound session? The initiating endpoint

first sets the destination, and it can be altered by any proxy or joinbox on its own authority. Also, on its own authority, any proxy can decide to be the accepting endpoint, so the session extends no further. On the way to each destination, the network infrastructure controls the routing and forwarding, which may steer the session packets through middleboxes. Just because a header in a compound session contains a particular destination name, the compound session is not guaranteed to reach a middlebox or accepting endpoint with that name; on the way to that destination, packets may be routed to a proxy or joinbox that changes the destination.

In a simple session, packets travel directly between the initiating and accepting endpoints, and each knows the identity (or has ways of knowing, see the chapter on *Network Security*) of the other. Obviously properties desired by initiating and accepting endpoints are more difficult to achieve in compound sessions. Because there are many possible session purposes and configurations, the solutions to this problem must currently be looked at on a case-by-case basis. For one example, the TCP session in Figure 14 satisfies both endpoints because the joinbox does not change the destination. If V is a client and $P2$ is a server, $P2$ probably does not care about the identity of the source, which *has* been changed.

Sidebars

Sidebar: Content Delivery Networks (CDNs)

Note: One interesting feature is that a cache for a Web site has its identity, keys, and certificate. This means that the same cache can have many identities.

Sidebar: The Accountable Internet

It is a fundamental characteristic of the classic Internet architecture that a member can send packets without being held accountable for them, because the source name in the packets can be false. This is consistent with the end-to-end principle, which implies that the cost of security measures such as endpoint authentication should be incurred only when they are needed, and paid only by those who need them. The consequence of the principle, for network security, is that attacks of all kinds are more difficult to counter because they cannot be blocked or prevented at their source.

The Accountable Internet is a proposed replacement for IP in which authentication of endpoints and networks ("accountability domains") has a primary role [2]. In the Accountable Internet Protocol (AIP) endpoint authentication is not implemented in user endpoints by session protocols, as is usual; rather it is part of routing and forwarding, and is implemented in AIP forwarders. Both endpoints and networks have self-certifying names derived from public keys, which eliminates the need for certificates binding identities to public keys. This is important in a global network, because there are no certificate authorities

that are trusted by all countries [8].

As a result, source names in packets are correct, and any packet can be attributed to the member that sent it. The costs are considerable and everyone connected to the Internet must bear them, which is why AIP is a radical proposal. The Accountable Internet's counter-argument to the end-to-end principle would be that endpoint authentication is essential for network security, so everyone needs it all the time.

In AIP the self-certifying name of a member is called an "endpoint identifier" (EID). An EID has three fields, the principal one being a 144-bit hash of the member's public key (note that an adequately secure public key must be at least 2000 bits long). In case the machine with an EID has multiple AIP interfaces, there are also 8 bits for distinguishing among different interfaces. There are also 8 bits for the version of cryptographic functions being used, so that the cryptographic scheme can evolve as older ones become easier to break.

In AIP every network ("accountability domain," AD) also has a public key and a self-certifying name derived from it. These have the same 160-bit format as EIDs, with the 8 interface bits all set to zero. AIP provides for large ADs that are hierarchically structured into smaller ADs, but we are omitting these details.

Mobility is built into AIP. In an AIP packet, the source and destination fields each contain an EID and the member's current location, which is an AD name. So an AIP DNS entry contains both EID and current AD, which the member must keep up-to-date. Long-distance routing and forwarding to a destination are based on its AD alone, while forwarding within an AD is based on the EID. Because the AD of an endpoint can change during a session, session protocols must incorporate session-location mobility, which means that there must be control messages that one endpoint can use to inform the other that it has moved to a new AD location.

For endpoint authentication (or "name certification"), The first forwarder encountered by an AIP packet with EID e sends back to the source a packet with a payload generated from the first packet, in a way that the forwarder keeps secret. The source is expected to encrypt this secret payload with the private key of e, and return it along with e's public key to the forwarder. If the forwarder can decrypt the encrypted secret payload with e's public key and get the original, and if e is indeed the 144-bit hash of the public key, then that packet source is accepted for a period of time as the legitimate endpoint with EID e. Other forwarders in the same AD could forward the packet if they trust that a previous forwarder has certified it, and if the forwarder's forwarding table specifies forwarding *to* the source EID on the same two-way link on which the packet arrived. Forwarders receiving a packet from a different AD could forward the packet if they trust that the AD it came from has certified it. Alternatively, any forwarder can repeat the challenge protocol.

The advantages of self-certifying names are even more important for networks than for endpoints, because self-certifying AD names are used to make inter-network routing (using BGP) secure. For secure IP routing, it is necessary to have a trusted association among three entities: AD name, public key, and

name prefixes owned by the AD. In AIP the AD name and public key are related algorithmically, and an AD's owned prefix *is* the AD name.

Many, and the most serious, DoS attacks are launched from botnets (see forthcoming sidebar). At first glance AIP is no defense against these attacks, because the attack packets bear their true source names. On the other hand, the machines of a botnet are almost always owned by innocent people who do not know that their machines have been compromised. So the AIP design includes a shut-off protocol to be implemented in smart network interface cards (NICs). To protect the NIC itself from penetration, it must be modifiable only through a dedicated physical interface, not from a network or from its host machine.

In the shut-off protocol, a member being attacked sends to each attacker (or at least some of them) a packet containing the victim's public key and a recent packet from the attacker encrypted with the victim's private key. The attacker's smart NIC can decrypt the packet with the public key, check that it recently sent the packet, and check that the victim's EID is the hash of the public key. If so, the NIC stops sending to the victim for a period of time specified in the packet. Without security in the shut-off protocol, the true attacker (master of the botnet) could exhaust the smart NIC's filtering state, so that it continues attacking the victim.

The AIP proposal introduces the inconvenient possibility that EIDs, which are computed from randomly chosen public keys, are not unique. The possible consequences of EID collision have not been elaborated.

Sidebar: Virtual Private Networks (VPNs)

Strictly speaking, VPNs are not a type of network, but rather a technology for widening the geographic span of a private IP network such as an enterprise network. Figure 15 shows the graph view of an enterprise network with members in two sites and on some remote laptops and home computers. VPN technology provides a secure implementation of the links that are not enclosed within sites, and therefore may cross the public Internet.

Figure 16 shows a session view of the enterprise network, in particular a TCP session between an employee laptop (currently located in a coffee shop) and an enterprise machine. The enterprise-network member on the laptop is described as a "VPN interface," because it is an IP interface plus VPN client. Before initiating the TCP session, it must first create a secure dynamic link to a VPN server in the enterprise network. It then sends packets of the TCP session over this link.

To create the dynamic link, the laptop's VPN interface requests that its IP interface make an IPsec session to public IP address *P4* (*P4* is part of its configuration). VPNs use IPsec in ESP Tunnel Mode with authentication, embedded in UDP for NAT traversal (§1.3, §1.4.2). The employee must also enter a password to authenticate his identity to the VPN server. The network name *V1.5* may be dynamic, and given to the VPN interface by the VPN server. After creation, the link transmits packets both ways with data encryption and message authentication.

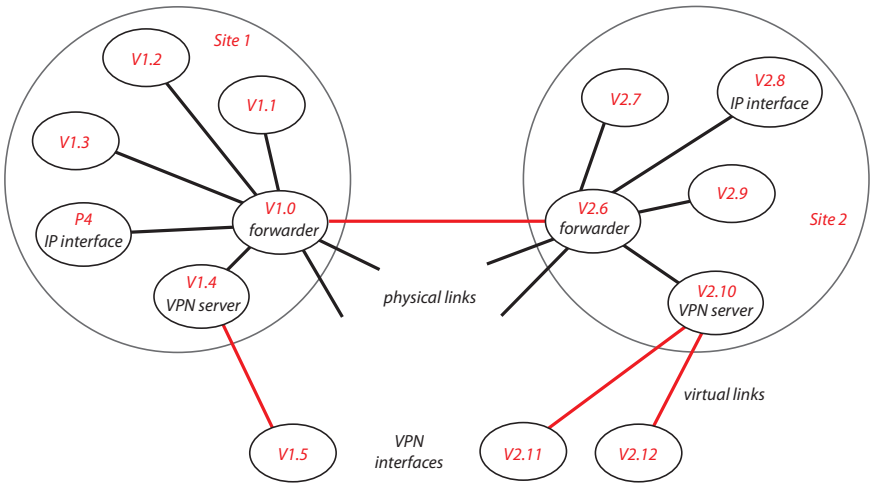

Figure 15: A graph view of an enterprise network using VPN technology. *V1* and *V2* are /24 prefixes of the private IP name space. *P4* is in the public IP name space. All links are two-way.

In Figure 16, the laptop's IP interface is connected to the public Internet through a WiFi network in the coffee shop. The WiFi network has only one public IP address (*P14*), which is the name of a combined firewall and NAT box protecting the network. It is the joinbox of a compound session, which is necessary because packets from *P4* cannot have the private IP address as their destination.

In Figure 15, the inter-site link between forwarders need not necessarily be implemented with IPsec. Another option is to get it from a particular service provider that offers "VPN service" to connect enterprise sites with dedicated virtual links through its network. If this link is implemented by a provider's VPN service, then it is based on whatever virtualization technology the service provider is using. In general the VPN concept has many possible implementations, of which IPsec is the most common.

Sidebar: Botnets

A botnet is not a network according to the strict definition in this book, but rather a group of user members of the public Internet. A botnet can be assembled by an illegal enterprise, and rented out to attackers for small fees.

A machine becomes part of a botnet when it is penetrated and infected with a particular kind of malware. This is all too easy for certain large classes of machines, which is why botnets of millions are common.

Machines are most easily penetrated when their manufacturers install de-

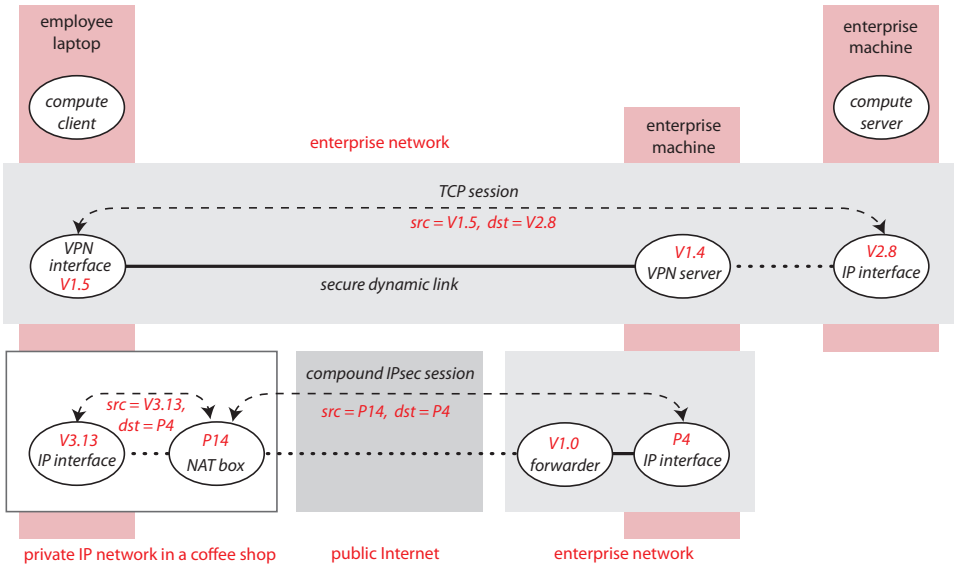

Figure 16: A session view of an enterprise network using VPN technology. A secure dynamic link in the enterprise network is implemented by a compound IPsec session traversing several bridged IP networks.

fault passwords in their software, and users connect them to the public Internet without changing their passwords. Attackers simply try IP addresses until they find such machines. Classes of such machines and their default credentials are posted on the Web for attackers to use.

Internet of Things machines are particularly vulnerable, because they include devices such as home routers, digital video recorders, and baby monitors often operated by users who are not computer-savvy. Another vulnerable class is virtual machines in public clouds, because they are initialized with operating-system defaults and can be penetrated before their customer has access to them!

Once a machine has become part of a botnet, it behaves normally until the master of the botnet sends it a triggering signal. Once triggered, it begins to generate IP packets to contribute to a flooding attack.

References

[1] D. G. Andersen. Mayday: Distributed filtering for Internet services. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, 2003.

[2] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). In *Proceedings of ACM SIGCOMM*, 2008.

[3] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. Netkat: Semantic foundations for networks. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, January 2014.

[4] K. Argyraki and D. R. Cheriton. Active Internet traffic filtering: Real-time response to denial-of-service attacks. In *Proceedings of the USENIX Annual Technical Conference*, 2005.

[5] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker. Don't mind the gap: Bridging network-wide objectives and device-level configurations. In *Proceedings of ACM SIGCOMM*, 2016.

[6] M. S. Blumenthal and D. D. Clark. Rethinking the design of the Internet: The end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology*, 1(1):70–109, August 2001.

[7] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *Proceedings of SIGCOMM*. ACM, August 2007.

[8] D. D. Clark. *Designing an Internet*. Information Policy Series, MIT Press, 2018.

[9] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: Defining tomorrow's Internet. *IEEE/ACM Transactions on Networking*, 13(3):462–475, June 2005.

[10] D. Dean, M. Franklin, and A. Stubblefield. An algebraic approach to IP traceback. *ACM Transactions on Information and System Security*, 5(2):119–137, May 2002.

[11] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[12] Dyn analysis summary of Friday October 21 attack. https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/. Accessed 10 November 2018.

[13] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: Validating SSL certificates in non-browser software. In *ACM Conference on Computer and Communications Security*, 2012.

[14] S. Goldwasser, Y. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proceedings of Symposium on Theory of Computing*. ACM, 2013.

[15] M. Handley. Why the Internet only just works. *BT Technology Journal*, 24(3):119–129, July 2006.

[16] How does TLS affect MQTT performance? https://www.hivemq.com/blog/how-does-tls-affect-mqtt-performance/. Accessed 19 September 2018.

[17] O. Honda, H. Ohsaki, M. Imase, M. Ishizuka, and J. Murayama. TCP over TCP: Effects of TCP tunneling on end-to-end throughput and latency. In *Proceedings of SPIE*, volume 6011, pages 138–146. International Society for Optical Engineering, 2005.

[18] A. Houmansadr, G. T. K. Nguyen, M. Caesar, and N. Borisov. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *Proceedings of the ACM Conference on Computer and Communications Security*, 2011.

[19] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. P. Mankins, and W. T. Strayer. Decoy routing: Toward unblockable Internet communication. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet*. USENIX, 2011.

[20] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte. Real time network policy checking using Header Space Analysis. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, 2013.

[21] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2012.

[22] S. Kent. IP encapsulating security payload (ESP). IETF Network Working Group Request for Comments 4303, 2005.

[23] A. D. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *Proceedings of SIGCOMM*. ACM, August 2002.

[24] S. Khattak, D. Fifield, S. Afroz, M. Javed, S. Sundaresan, V. Paxson, S. J. Murdoch, and D. McCoy. Do you see what I see? Differential treatment of anonymous users. In *Proceedings of the Network and Distributed Security Symposium*. Internet Society, 2016.

[25] T. Kiravuo, M. Sarela, and J. Manner. A survey of Ethernet LAN security. *IEEE Communications Surveys & Tutorials*, 15(3):1477–1491, 2013.

[26] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach, Seventh Edition*. Pearson Education, Inc., 2017.

[27] N. P. Lopes, N. Bjorner, P. Godefroid, K. Jayaraman, and G. Varghese. Checking beliefs in dynamic networks. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, 2015.

[28] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Schenker. Controlling high bandwidth aggregates in the network. *Computer Communication Review*, 32(3):62–73, July 2002.

[29] G. C. M. Moura, J. Heidemann, M. Muller, R. de O. Schmidt, and M. Davids. When the dike breaks: Dissecting DNS defenses during DDoS. In *Proceedings of the ACM Internet Measurement Conference*, 2018.

[30] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafo, K. Papagiannaki, and P. Steenkiste. The cost of the 'S' in HTTPS. In *Proceedings of ACM CoNEXT*, 2014.

[31] D. Naylor, R. Li, C. Gkantsidis, , T. Karagiannis, and P. Steenkiste. And then there were more: Secure communication for more than two parties. In *Proceedings of ACM CoNEXT*, 2017.

[32] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. Lopez, K. Papagiannaki, P. R. Rodriguez, and P. Steenkiste. Multi-context TLS (mcTLS): Enabling secure in-network functionality in TLS. In *Proceedings of ACM SIGCOMM*, 2015.

[33] Y. Sheffer, R. Holz, and P. Saint-Andre. Summarizing known attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS). Internet Engineering Task Force Request for Comments 7457, 2015.

[34] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. BlindBox: Deep packet inspection over encrypted traffic. In *Proceedings of SIGCOMM*. ACM, 2015.

[35] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman. Telex: Anticensorship in the network infrastructure. In *Proceedings of the 20th USENIX Security Symposium*, 2011.

[36] A. Yaar, A. Perrig, and D. Song. SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 2004.

[37] X. Yang, D. Wetherall, and T. Anderson. A DoS-limiting network architecture. In *Proceedings of SIGCOMM*. ACM, August 2005.

[38] P. Zave and J. Rexford. The compositional architecture of the Internet. 2018, to appear in *Communications of the ACM*.