6.857 Lecture: Network Security

November 15, 2005

Readings:

-

In this lecture:

- The layered model; vulnerabilities at each layer
- Network security standards: certificates, SSL/TLS, IPsec, IKE
- Firewalls

1 The Layered Model

Recall from 6.033 that networks are abstracted into a "layered model," where each layer provides an abstract interface that can be stacked on top of many different types of lower layers. Using a coarse separation, these layers can be described as the following (from lowest- to highest-level):

1. Link or "media access control" (MAC) layer; e.g., ethernet, token ring, wireless
2. Network layer; e.g., Internet Protocol (IP)
3. Transport layer; e.g., Transmission Control Protocol (TCP), Universal Datagram Protocol (UDP)
4. End-to-end or application layer; e.g., SSH, HTTP

We will briefly review the function of each layer, and talk about the vulnerabilities that should be addressed when building a secure system.

1.1 Link layer

The link layer is responsible for moving bits from one place to another over a *single* link. This link is usually a physical medium, and hence of an analog nature. Ethernet, for example, specifies how bits should be encoded as voltages on a wire, and how to deal with collisions (more than one link endpoint attempting to send at the same time).

The security of the link layer is usually a question of physical security of the actual link medium: can an ethernet wire be cut or spliced? Can wireless frequencies be jammed? How easy is it to become a link endpoint? What are the consequences of using a broadcast link versus pairwise links?

One example of a security exploit at the link layer is a *packet sniffer* on an Ethernet network. Ethernet is a broadcast medium, so any host on the network can in principle see all the packets sent and received over the wire. Packet sniffing software (e.g., `ethereal`) puts the Ethernet card in "promiscuous mode," which causes the card to report every packet to the operating system. This can be used to eavesdrop on any data that is sent in the clear, which may include passwords, email traffic, web pages, or other sensitive information. (Packet sniffers are also used for many legitimate reasons, such as debugging network software.) Packet sniffing can also be done on wireless networks, which can be even easier to listen in on, given that there is no need to physically "plug in" to a network port.

1.2 Network layer

The network layer is responsible solely for moving packets of data from one location to another, where the locations may be spanned by several links. Therefore the network layer is concerned chiefly with examining the addresses of the endpoints, and routing packets over several links in an attempt to deliver a packet to its destination address. It does not provide any guarantees about delivery, acknowledgement of receipt, etc.

The security of the network layer is generally a question of the security of the addressing and routing system: can routers be accidentally or maliciously misconfigured? Can a source address be spoofed? Can the address-lookup procedure be compromised?

In IP, an attack called "IP spoofing" is very easy. It stems from the fact that routers are usually only concerned with delivering a packet to its destination; they typically don't look at the source IP address in the packet header. A malicious source host can therefore declare *any* IP address, the router will deliver it the packet, and the destination host will believe that it came from the spoofed source.

It's not immediately clear why this attack is useful — after all, if the destination host replies to the declared source address, the attacker will probably not receive that reply. However, in many applications there may be subtle security flaws if one incorrectly assumes that the source address in the IP header is correct (we'll see one example below).

IP spoofing must be prevented near the source, and largely depends on networks acting as "good neighbors." Routers at the border of a network should check that the source IP address of each outgoing packet belongs to the router's internal network. This prevents spoofing attacks from originating from *within* the router's network, but it doesn't do anything

to prevent attacks from outside. Also, the router can check that the source address of each *incoming* packet is *not* from its internal network. This might filter out a few attacks coming from outside the router's network, but it cannot stop all of them (spoofed addresses can still be from outside the attacked network).

[maybe BGP security, maybe DNS poisoning]

1.3 Transport layer

The transport layer is responsible for providing some guarantees about a communication stream between two systems on the network. Such guarantees might be: packets are received in the order in which they are sent, every packet will attempt to be delivered until it is received, packets will be acknowledged when they are delivered, etc. TCP provides guarantees like these.

The transport layer can contain a lot of complexity, and hence it may be difficult to reason about its security. For example, the transport layer might need to guard against "session hijacking," in which an attacker inserts well-crafted bad packets into a connection, causing the bogus packets to be accepted and the legitimate ones to be ignored.

Here is a well-known attack that exploits both the network and transport layers, called a "SYN flood." First, some background: every TCP session starts off with a "handshake" protocol, to ensure that both sides of the connection can access each other. The handshake goes as follows:

1. Client sends Server a SYN message.
2. Server sends Client a SYN-ACK message.
3. Client responds with an ACK message.
4. Client and Server exchange service-specific data.

A basic SYN attack works as follows: a client sends a SYN message, the server replies with SYN-ACK, but the client never responds. At this point the connection is "half-open," because it is open from the server's point of view, and the server will not time-out the connection for several seconds. The server maintains in its memory a data structure describing the set of all open connections, and this structure is of a finite size. The attack consists of flooding the server with thousands of SYN messages, causing the data structure to overflow (potentially crashing the machine) or leaving the server unable to respond to any legitimate new TCP connections when the set fills up.

As the attack has been described so far, there is an easy fix: the server can just ignore new TCP connections from the attacking client, past a certain number of open sessions. However, we saw above that a malicious client can use IP spoofing to hide its identity. Note that the SYN flood attack doesn't require the client to respond to any of the server's messages, so if the client uses IP spoofing, the server is helpless.

There is no silver bullet to fix this problem; it is a deficiency in the design of TCP. In Linux, a good countermeasure called "SYN cookies" works as follows: when the set of

open connections is nearly full, instead of replying with ACK, the server replies with a SYN "cookie," and removes the SYN from the set of open connections. Cookies expire very quickly. If value of the cookie is echoed back in time, the TCP connection is considered open, otherwise the cookie is forgotten.

The cookie essentially decreases the time-out threshold, but *only* when the server is under attack. (Under normal conditions, the client will have plenty of time to ACK.) Also, if an attacker is using IP spoofing, the SYN cookie will be delivered to the fake IP address and not the attacker, making the attacker unable to correctly reply to the cookie value.

[maybe RST attacks?]

1.4 Application layer

The application layer is a protocol for transmitting data in a format that a particular application can understand. For example, HTTP defines a language specifying how a web browser should request pages from a server, and how the server should reply with the pages' contents.

The application layer is generally the source of greatest complexity. Many application protocols, such as email, have not been designed with security in mind. The consequences have included: forgeable source addresses, spam email (where the bulk of delivery effort is not performed by the actual sender), and email worms. Throughout the course we will see many examples of security failures in different network application protocols.

1.5 End-to-end arguments

When deciding which layers should implement which functions, the "end-to-end argument" in network protocol design is a good rule of thumb. It discourages placing features at lower layers, because those features will probably not be exactly what the application needs — either because those needs are impossible to predict precisely, or because the lower levels do not have access to the proper abstractions. Therefore the application will either (1) re-implement the exact feature it needs, or (2) incorrectly assume that the required feature is actually provided by the lower layer. In the former case, the lower layers lose efficiency by implementing unused features; in the latter case, the application may suffer from bugs and/or security flaws. (We have already seen how incorrect assumptions about source address authenticity in IP caused a security flaw in TCP.)

End-to-end arguments can also be made for security, and are sometimes even more compelling than just the correctness and efficiency arguments. Consider the example of a secure (encrypted and authenticated) channel between a web browser and web server, and suppose the browser is connected to the Internet via a wireless network. WEP is a widely-implemented standard for encrypting and authenticating packets at the wireless link layer, between the network card and the wireless access point. There are several problems with relying on WEP for this application:

- Once the data reaches the access point, it is decrypted and travels over the internet in the clear, where it is vulnerable to eavesdropping.
- The access point has no means to authenticate the web server to the browser, nor vice-versa. (The keys of these parties lie at a higher layer of abstaction than the link layer.)
- WEP only implements a limited set of encryption and authentication protocols, and they have serious design flaws (weak encryption, terrible authentication).

Therefore, relying on WEP only gives a false sense of security for this application. A much better solution is for the browser and server to implement SSL (secure sockets layer), which provides an end-to-end secure transmission channel at a level above the TCP transport layer. This has its own problems — for example, by inserting bogus packets at the TCP layer (which is oblivious to the SSL connection, and cannot filter out these inauthentic packets), an adversary can deny service at the SSL layer. But the security of the connection is not compromised.

This is not to say that WEP is useless — for example, it can still authenticate the wireless card to the access point, to prevent malicious connections or free-loaders. But this is a problem that resides at the link layer, and the link layer is uniquely positioned to solve it.

2 Network Security Standards

There are a number of standards aimed at securing different pieces of the Internet infrastructure. They reside at different layers in the protocol stack and perform different functions, giving each one distinct advantages and disadvantages for different applications.

2.1 Certificates

In the world of public-key cryptography, there is always the troubling question: how does Alice know that a given public key really belongs to Bob? After all, anyone can publish a public key and claim it belongs to anyone, and a man-in-the-middle can always prevent Alice from getting a copy of Bob's actual public key.

Certificates are one way to solve this problem. A certificate is a way to bind a public key to an identity. The most widely-deployed certificates employ one or more trusted third parties called *certificate authorities* (CAs). We assume that Alice knows the actual public key pk_{CA} of some CA; usually it comes as part of her encryption software. (Real-world CAs include VeriSign, Thawte, RSA Data Security, and many others.)

The certification process works as follows:

1. Bob generates a fresh key pair (sk_B, pk_B).

2. Bob goes to the certificate authority with his public key pk_B, and proves to the CA that he really is Bob (usually with some kind of legal documents like a birth certificate).

3. The CA, using its secret key sk_{CA}, digital signs the statement of the form: "pk_B belongs to Bob." This statement, along with the CA's signature, is the certificate. Bob publishes the certificate and/or delivers it to anyone upon request.

When Alice wants to encrypt a message to Bob, she first obtains a copy of Bob's certificate. Using pk_{CA}, she verifies the digital signature in the certificate. Because she trusts the CA to check identities properly, she now believes that pk_B belongs to Bob. Now she can use pk_B to encrypt messages to Bob.

In the real world, the actual contents of a certificate are a bit more complex. The notion of an "identity" on the Internet can be a bit slippery. For e-commerce, banking, and related applications, identity includes a website domain name (e.g., `amazon.com`), as well as a real-world organization name (e.g., Amazon.com Inc.), and a range of valid dates for the identity. Certificate authorities need to verify that all of this information is correct before issuing a certificate.

Certificates also include information about the public key's type (e.g., RSA, ElGamal), whether it should be used for encryption or signature verification, how to check if the key has been revoked by the owner, and more. A standard called X.509 is the most widely-used way of specifying and formatting this information and verifying the CA's signature.

2.2 SSL/TLS

Secure sockets layer (SSL) is implemented at a layer above TCP, i.e. at the application level. This allows it to be used without making changes to the operating system, and without drastic changes to existing applications. It provides an encrypted, authenticated (to one side, anyway) channel between two hosts.

[Historical note: TLS (Transport Layer Security) is a tweaked version of SSL version 3, which was created for the purposes of having a patent-unencumbered, open standard. However, it is incompatible with SSLv3. Now that's SSL's patents have expired, it's doubtful that TLS will catch on too widely.]

The goal of SSL is to compute some shared *session keys* that are known to both Alice and Bob, but nobody else. These session keys are used to efficiently encrypt and authenticate all the messages sent between Alice and Bob for the remainder of the session (say, using AES and a MAC). Additionally, SSL provides a way for Bob to authenticate himself to Alice (and, optionally, vice-versa), and for Alice and Bob to be assured that there is no man-in-the-middle between them.

We assume that Bob has a certificate that has been issued by a CA that Alice trusts. Here is an outline of the SSL protocol preamble:

1. Alice $\rightarrow$ Bob: Alice contacts Bob (but does not identify herself), gives a list of her supported crypto algorithms, and a random number R_A.

2. Bob → Alice: Bob sends Alice his certificate, a random number R_B, and tells Alice which of her supported ciphers to use.

3. Alice → Bob: Alice chooses a random S (called the *pre-master secret*), encrypts it under Bob's public key (taken from his certificate), and sends it to Bob. Formally, Alice sends $E_{pk_B}(S)$ to Bob.

   Alice also computes the *master secret* K, which is derived from S, R_A, R_B — this computation is expensive, using public-key operations. All the session keys will be derived from K using hashing (the details aren't important).

   Alice also uses K to compute a "keyed hash" of the previous handshake messages, and sends it to Bob. This serves to prove to Bob that she knows K, and that the handshake messages were not tampered with by a man-in-the-middle.

4. Bob → Alice: Bob computes a keyed hash of all prior handshake messages, encrypts and authenticates it with the proper session keys, and sends it to Alice. This ensures that the handshake messages were not tampered with, and proves that Bob knows the secret key associated with his public key — otherwise he could not have decrypted S, computed K, nor derived any of the session keys.

At this point, Alice knows she is talking to Bob, but Bob doesn't know to whom he's talking. Extensions to the protocol allow Alice to authenticate herself using a certificate. More commonly, Alice can authenticate herself by filling in a web form with her username and password, which will be encrypted via the established SSL channel.

2.3 IPsec

In contrast with SSL, IPsec ("IP security") is implemented at the network layer.[1] It defines extensions to IP headers to support authenticity, encryption, or both. It assumes that the two endpoint hosts have already established a shared secret session key (via IKE, typically).

The standard for authenticity-only transmission is called AH (for Authentication Header); for encryption and/or authenticity, it is called ESP (Encapsulating Security Payload). ESP (almost) provides a superset of the functionality of AH, but AH continues to exist mainly because of political reasons (within standards committees, and also due to governmental export restrictions on encryption software).

The one difference between authenticity in ESP and AH is that AH provides integrity of some of the fields inside the actual IP header, and ESP does not. It's not clear what this is useful for, though.

When ESP is used, intermediate routers and firewalls cannot examine the contents of the packets, e.g. to filter out traffic based on TCP port, because those packets may be encrypted using keys that only the endpoints know. Firewall administrators probably don't like this property, but users might.

[1] By definition, it's design forgoes the end-to-end argument.

Some distinguishing between Tunnel and Transport modes of IPsec?

There are also more technical issues with NAT and firewalls using each of these modes. What a disaster.

IKE is comlpetely incomprehensible, due to politics in the standards board and desire for the most flexibility within the protocol.