Information Security: Principles and Practice, Second Edition
by Mark Stamp

APPENDIX

This appendix includes two sections. The first section contains an abbreviated introduction to networking, with the emphasis on security issues. The second section provides a quick review of the basic math that is used in various parts of this book.

A-1 Network Security Basics

There are three kinds of death in this world.
There's heart death, there's brain death, and there's being off the network.
— Guy Almes

A-1.1 Introduction

In this section, we give a condensed introduction to networking, presented through the prism of security. Networking is a large and complex topic. Here, we'll cover the minimal amount of information that is required elsewhere in this textbook, and we'll also add a few passing comments on network-specific security issues that are of independent interest.

A network consists of *hosts* and *routers*. The term host is a catchall for a wide variety of network-connected devices, including laptops, desktop computers, servers, cell phones, PDAs, etc. The purpose of the network is to transfer data between the hosts. Ideally, we'd like the network to be transparent to users. We're primarily concerned with the mother of all networks, the Internet.[1]

A network has an *edge* and a *core*. The hosts mentioned above live at the edge, while the core consists of an interconnected mesh of routers. The purpose of the core is to route data through the network from host to host. A generic network diagram appears in Figure A-1.

[1] And, of course, everyone knows that the Internet was invented by Al Gore.

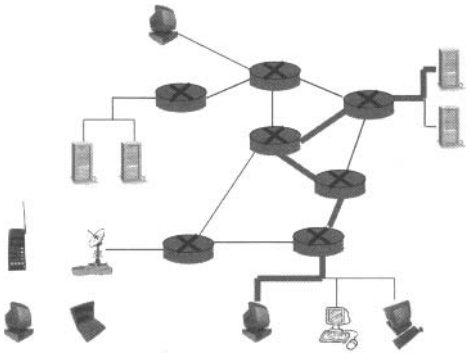

Figure A-1: Network

The Internet is a *packet switched* network, meaning that the data is sent in discrete chunks known as packets. In contrast, the traditional telephone system is a *circuit switched* network. For each telephone call, a dedicated circuit—with dedicated bandwidth—is established between the end points. Packet switched networks can make more efficient use of the available bandwidth, although there is some additional complexity, and things get particularly involved if circuit switched-like behavior is desired.

The study of modern networking is largely the study of networking protocols. Networking protocols precisely specify communication rules employed by the network. For the Internet, the details are usually spelled out in RFCs, which are, in effect, Internet standards.[2]

Protocols can be classified in many different ways, but one classification that is particularly relevant in security is *stateless* versus *stateful*. Stateless protocols don't "remember" anything, while stateful protocols do have some "memory." Many security problems are related to state. For example, denial of service, or DoS, attacks often take advantage of stateful protocols, while stateless protocols can also have their own security issues, as we'll see below.

A-1.2 The Protocol Stack

It's standard practice to view networks in terms of layers, where each layer is responsible for some particular operations. When these layers are all stacked up, the result is, not surprisingly, known as a *protocol stack*. It's important

[2]RFC stands for Request for Comments. However, authors of RFCs are not actually requesting comments. Instead, RFCs act as Internet standards. But curiously, most RFCs are not official Internet standards and, in fact, only a relatively few RFCs have been promoted to the level of official Internet standards. How does a lowly RFC become a high-falutin' Internet standard? Well, it's all spelled out in in RFC 2026, which is itself not an Internet standard. Confused?

to realize that a protocol stack is more conceptual than an actual physical construct. Nevertheless, the idea of a protocol stack does simplify the study of networks—although newcomers to networking are excused for not believing it. The infamous OSI reference model includes seven layers, but we'll strip it down to the layers that matter, which only leaves the following five:

- The *application layer* is responsible for handling the application data that is sent from host to host. Examples of application layer protocols include HTTP, SMTP, FTP, and Gnutella.
- The *transport layer* deals with logical end-to-end transport of the data. The transport layer protocols of interest are TCP and UDP.
- The *network layer* is responsible for routing data through the network. IP is the network layer protocol that matters most to us.
- The *link layer* handles the transferring of data over individual links within the network. There are many link layer protocols, but we'll only mention two, Ethernet and ARP.
- The *physical layer* sends the bits over the physical media. If you want to know about the physical layer, take an electrical engineering course.

Conceptually, a packet of data passes down the protocol stack (from the application layer to the physical layer) at the source and then back up the protocol stack at the destination. Routers in the core of the network must process packets up to the network layer so they can make sensible routing decisions. Layering is illustrated in Figure A-2.

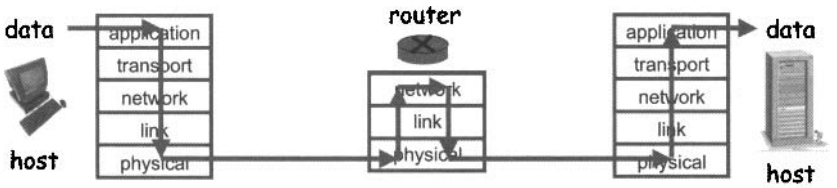

Figure A-2: Layering in Action

Suppose that X is a freshly minted packet of application data. As X goes down protocol stack, each protocol adds additional information, usually in the form of a *header*, which includes information required by the protocol being used at that particular layer. Let H_A be the header added at the application layer. Then the application layer passes (H_A, X) down the stack to the transport layer. If H_T is the transport layer header, then $(H_T, (H_A, X))$ is passed to the network layer where another header, say, H_N is added to

give $(H_N, (H_T, (H_A, X)))$. Finally, the link layer adds a header, H_L, and the packet

$$(H_L, (H_N, (H_T, (H_A, X))))$$

is passed to the physical layer. In particular, note that the application layer header is the innermost header, which might seem backward until you think about it a little bit. When the packet is processed up the protocol stack at the destination (or at a router), the headers are stripped off layer by layer—like peeling an onion—and the information in each header is used to determine the proper course of action by the corresponding protocol.

Next, we'll take a brief look at each of the layers. We'll follow [177] and go down the protocol stack from the application layer to the link layer.

A-1.3 Application Layer

Typical network applications include Web browsing, email, file transfer, P2P, and so on. These are distributed applications that run on hosts. The hosts would prefer the network to be completely transparent.

As mentioned above, HTTP, SMTP, IMAP, FTP, and Gnutella are examples of application layer protocols. Note that the protocol is only one part of an application. For example, an email application includes an email client (such as Outlook or Thunderbird), a sending host, a receiving host, email servers, and various networking protocols such as SMTP and POP3.

Most applications are designed for the client-server paradigm, where the *client* is the host that requests a service and the *server* is the host that responds to the request. In other words, the client is the one who speaks first and the server is the one trying to fulfill the request. For example, if you request a Web page, you are the client and the Web server is the server, which only seems right. However, in some cases the distinction between client and server is not so obvious. For example, in a file-sharing application, your computer is a client when you request a file, and it is a server when someone downloads a file from you. Both of these events could even occur simultaneously, in which case you would be both a client and a server at the same time.

Peer-to-peer, or P2P, file sharing applications offer something of an alternative to the traditional client-server model. In the P2P model, hosts act as both clients and servers, as mentioned in the previous paragraph. But the real challenge in P2P lies in locating a "server" with the content that a client desires. There are several interesting approaches to this problem. For example, some P2P systems distribute the database that maps available content to hosts among certain special peers, whereas others simply flood each request through the network. In the latter case, hosts with the desired content respond directly to the requester. For example, KaZaA uses the distributed database approach, while Gnutella employs query flooding.

Next, we'll briefly discuss a few specific application layer protocols. First, let's consider HTTP, the HyperText Transfer Protocol, which is the application layer protocol used when you browse the Web. As mentioned above, the client requests a Web page and the server responds to the request. Since HTTP is a stateless protocol, *Web cookies* were developed as a tasty way to maintain state. When you initially contact a Web site, the Web site can choose to provide your browser with a cookie (assuming your browser is willing to accept it). A cookie is simply an identifier that is used to index a database maintained by the Web server. When your browser subsequently sends HTTP messages to the Web server, your browser will automatically pass the cookie to the server. The server can then consult its database and thereby remember information about you. In this way, Web cookies make it possible to maintain state within a single session as well as across sessions.

Web cookies are also sometimes used as a very weak form of authentication and cookies enable such modern conveniences as shopping carts and recommendation lists. However, cookies do raise some privacy concerns, since a Web site with memory (which is enabled by cookies) can learn a great deal about you. This problem only gets worse if multiple sites pool their information, since they can probably gain a fairly complete picture of your Web persona.

Another interesting application layer protocol is SMTP, the Simple Mail Transfer Protocol, which is used to transfer email from the sender to the recipient's email server. Then POP3, IMAP, or HTTP (for Web mail) is used to transfer the messages from the email server to the recipient. An SMTP email server can act as a server or a client when email is transferred over the network.

As with many application protocols, SMTP commands are human readable. For example, the commands in Table A-1 are legitimate SMTP commands that were typed as part of a telnet session—the user typed the lines beginning with `C` while the server responded with the lines marked as `S`. This particular session resulted in a spoofed email being sent to your gullible author at `stamp@cs.sjsu.edu` from `arnold@ca.gov`.

Another application layer protocol with security implications is DNS, the Domain Name Service. The primary purpose of DNS is to convert a friendly human-readable name, such as `www.google.com`, into its equivalent 32-bit IP address (discussed below), which computers and routers prefer. DNS is implemented as a distributed heirarchical database. There are only 13 "root" DNS servers worldwide and a successful attack on these would cripple the Internet. This is perhaps as close to a single point of failure as exits in the Internet today. Attacks on root servers have succeeded, however, because of the distributed nature of the DNS, it would be necessary for such an attack to continue for an extended period of time before it would seriously affect the Internet. No attack on DNS has had such staying power—at least not yet.

Table A-1: Spoofed email in SMTP

```
C: telnet eniac.cs.sjsu.edu 25
S: 220 eniac.sjsu.edu
C: HELO ca.gov
S: 250 Hello ca.gov, pleased to meet you
C: MAIL FROM: <arnold@ca.gov>
S: 250 arnold@ca.gov... Sender ok
C: RCPT TO: <stamp@cs.sjsu.edu>
S: 250 stamp@cs.sjsu.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: It is my pleasure to inform you that you
C: are terminated
C:  .
S: 250 Message accepted for delivery
C: QUIT
S: 221 eniac.sjsu.edu closing connection
```

A-1.4 Transport Layer

The network layer (discussed below) offers unreliable, "best effort" delivery of packets. This means that the network layer attempts to get packets to their destination, but if a packet fails to arrive (or its data is corrupted or a packet arrives out of order or ...), the network takes no responsibility, much like the U.S. Postal Service. Any improved service beyond this limited best effort—such as the reliable delivery of packets—must be implemented somewhere above the network layer. Also, such additional service must be implemented on the hosts, since the core of the network only offers this best-effort delivery service. Reliable delivery of packets is the primary purpose of the transport layer.

Before we dive into the transport layer it's worth pondering why the network layer is allowed to be unreliable by design. Recall that we are dealing with a packet switched network. Consequently, it's possible that hosts will put more packets into the network than it can handle. Routers include buffers to store extra packets until they can be forwarded, but these buffers are finite—when a router's buffer is full, the router has no choice but to drop packets. The data in packets can also get corrupted in transit. And, since routing is a dynamic process, it's possible that packets in one particular connection can follow different paths. When this occurs, the packets can arrive at the destination in a different order than they were sent by the source. It's the job of the transport layer to deal with such reliability issues. The bottom line is that routing packets through the core of the network is difficult, so the

designers of the Internet decided to minimize the burden at this level, and thus the minimal best effort approach at the network layer.

There are two transport layer protocols of importance: TCP and UDP. The Transmission Control Protocol, or TCP, provides for reliable delivery. TCP will make sure that your packets arrive, that they are sequenced in the correct order, and that the data has not been corrupted. To oversimplify things, the way that TCP provides these services is by including sequence numbers in packets and telling the sender to retransmit packets when problems are detected. Note that TCP runs on hosts, and all communication is over the same (unreliable) network where the data is sent. The format of the TCP header appears in Figure A-3.

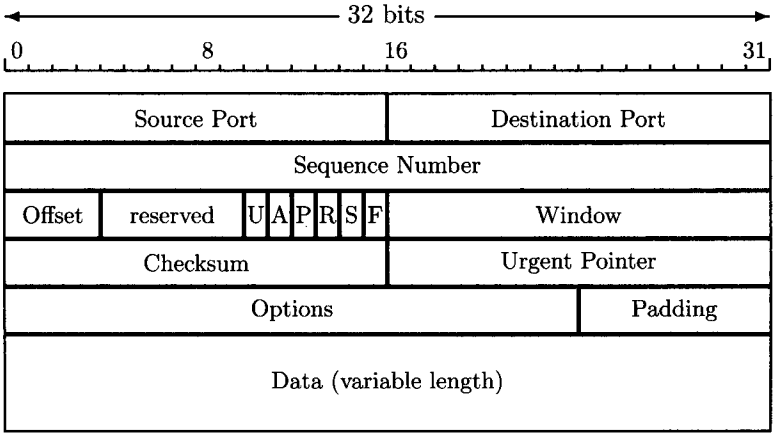

Figure A-3: TCP Header

TCP assures that packets arrive at their destination and that they are processed in order. TCP also makes sure that packets are not sent too fast for the receiver, which is known as *flow control*. In addition, TCP provides network-wide *congestion control*. This congestion control feature is complex, but one interesting aspect is that it attempts to give every host a fair share of the available bandwidth. That is, if congestion is detected, every TCP connection will get about the same amount of the available bandwidth. Of course, everyone wants more than their fair share, so hosts can (and do) try to cheat this congestion control feature by opening multiple TCP connections.

TCP is said to be connection-oriented, which means that TCP contacts the server before sending data. That is, TCP checks that the destination server is alive and listening on the appropriate port. It's important to realize that this TCP "connection" is only a logical connection—no true dedicated connection takes place.

The TCP connection establishment is of particular importance. A so-called *three-way handshake* is used, where the three messages that are exchanged are the following:

- SYN — The client requests "synchronization" with the server.
- SYN-ACK — The server acknowledges receipt of the SYN request.
- ACK — The client acknowledges the SYN-ACK. This third message can also include data. For example, if the client is Web browsing, the client could include the request for a specific Web page along with the ACK message.

The three-way handshake is illustrated in Figure A-4.

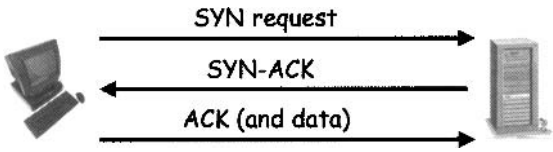

Figure A-4: TCP 3-Way Handshake

TCP also provides for orderly tearing down of connections. Connections are terminated by a process involving a FIN (finish) packet or by a single RST (reset) packet.

The TCP three-way handshake makes denial of service, or DoS, attacks possible. Whenever a SYN packet is received, the server must remember the so-called "half-open" connection. This remembering consumes a small amount of server resources. As a result, too many half-open connections will cause server resources to be exhausted, at which point the server can no longer respond to new connections.

A straightforward DoS attack that is launched from a single machine using a single IP address is relatively easy to defend against—the intended victim can simply ignore or block any IP address that sends too many TCP requests. The attacker could make the attack difficult to block by spoofing the source IP addresses to make it appear that the requests are coming from many different machines. However, the amount of traffic needed to significantly affect the victim is likely to be more than one machine can generate. Consequently, most successful DoS attacks are actually distributed denial of service, or DDoS, attack. In a DDoS attack, many different machines are used to overwhelm the victim. If a large number of machines are used in a DDoS attack, then the generated traffic may be sufficient to prevent the victim from responding to legitimate requests. The distributed nature of such an attack makes it difficult to defend against.

The transport layer includes another protocol of note, the User Datagram Protocol, or UDP. Whereas TCP provides everything and the kitchen sink, UDP is a truly minimal no-frills service. The benefit of UDP is that it requires minimal overhead, but the tradeoff is that it provides no assurance that packets arrive, no assurance packets are in the proper order, and so on. In other words, UDP adds little to the unreliable network over which it operates.

Why does UDP exist? UDP is more efficient since it has a smaller header, but the major potential benefit derives from the fact that UDP has no flow control or congestion control. Due to the lack of these controls, there are no restrictions to slow down the sender. However, if packets are sent too fast, they will be dropped—either at an intermediate router or at the destination. So, how can UDP be a good thing? In some applications, delay is not tolerable, but it is acceptable to lose some fraction of the packets. Streaming audio and video fit this description, and for these applications UDP is generally preferable to TCP. In effect, UDP allows an application to get more than its fair share of the bandwidth, at the risk of packets getting dropped. Finally, it's worth noting that reliable data transfer over UDP is possible, but the reliability must be built in by the developer at the application layer. This would seem to provide the best of both worlds—reliability with no bandwidth limitations—at the expense of a more complex application layer protocol.

A-1.5 Network Layer

The network layer is the crucial layer for the core of network. Recall that the core is an interconnected mesh of routers, and the purpose of the network layer is to provide the information needed to route packets through this mesh. The network layer protocol of interest here is the Internet Protocol, or IP. As mentioned above, IP follows a best effort approach. Note that IP must run in every host and router in the network. The format of the IP header appears in Figure A-5.

In addition to network layer protocols, routers also run routing protocols. The purpose of a routing protocol is to determine the best path to use when sending a packet. There are many routing protocols, but the most popular are RIP, OSPF, and BGP. These protocols are very interesting, but we won't discuss them here.

Every host on the Internet must be associated with a 32-bit IP address. Unfortunately, there are not enough IP addresses for the number of hosts, and as a result many tricks are employed to effectively extend the IP address space. IP addresses are given in so-called dotted decimal notation of the form $W.X.Y.Z$, where each value is between 0 and 255. For example, 195.72.180.27 is a valid IP address. Note that a host's IP address can—and often does—change.

Although each host has a 32-bit IP address, there can be many processes

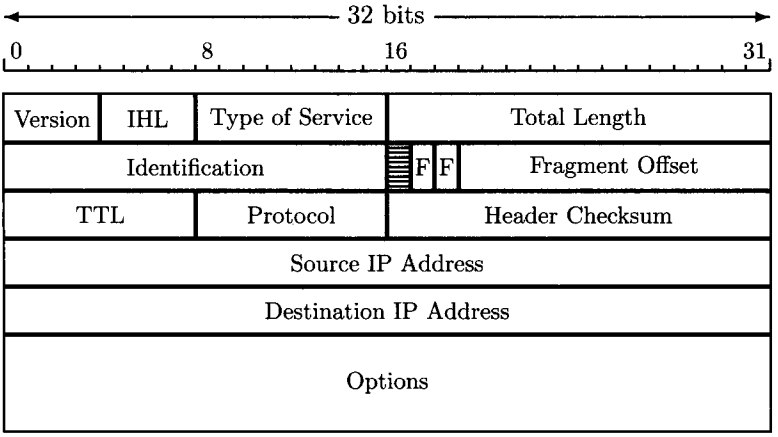

Figure A-5: IP Header

running on a single host. For example, you could browse the Web, send email, and do a file transfer all at the same time. To effectively communicate across the network, it's necessary to distinguish these processes. The way this is accomplished is by assigning each process a 16-bit *port number*. The port numbers below 1024 are said to be well known, and they're reserved for specific applications. For example, port 80 is used for HTTP and port 110 is for POP3. The port numbers from 1024 to 65535 are dynamic and assigned as needed. An IP address together with a port number defines a *socket*, and a socket uniquely identifies a process on the Internet.

The IP header is used by routers to determine the proper route for a packet through the network. The header includes fields for the source and destination IP addresses. There is also a time-to-live, or TTL, field that limits the number of hops that a packet can travel before it dies and goes to packet heaven. This prevents wayward packets from bouncing around the Internet for all of eternity. There are also fields that deal with fragmentation, which is our next topic.

Each link on the Internet limits the maximum size of packets. If a packet is too big, it's the router's job to split it into smaller packets. This process in known as *fragmentation*. To prevent multiple fragmentation and reassembly steps, the fragments are only reassembled at the their destination.

Fragmentation creates many security issues. One problem is that the actual purpose of a packet is easily disguised by breaking it into fragments. The fragments can be arranged to overlap when reassembled, which further exacerbates this problem. The result is that the receiving host can only determine the purpose of a packet after it has received all of the fragments

and reassembled the pieces. A firewall has a great deal more work to do when dealing with fragmented packets. As a result, fragmentation opens the door to DoS and many other types of attacks.

Currently, we use IP version 4, that is, IPv4. It has many shortcomings, including too-small 32-bit addresses and poor security (fragmentation being just one example). As a result, a new-and-improved version, IP version 6 (IPv6), has been developed. IPv6 includes 128-bit addresses—which gives a virtually inexhaustible supply of IP addresses—and strong security in the form of IPSec. Unfortunately, IPv6 is a classic example of how not to develop a replacement protocol. There is no natural way to migrate from IPv4 to IPv6 and, consequently, IPv6 has yet to take hold on a large scale [30].

A-1.6 Link Layer

The link layer is responsible for getting the packet over each individual link in the network. That is, the link layer deals with getting a packet from a host to a router, from a router to a router, from a router to a host or, locally, from one host to another host. As a packet traverses the network, different links can be completely different. For example, a single packet might travel over Ethernet, a wired point-to-point line, and a wireless microwave link when traveling from its source to its destination.

In each host, the link layer and physical layer are implemented in a semi-autonomous adapter known as a Network Interface Card, or NIC—examples include Ethernet cards and wireless 802.11 cards. The NIC is (mostly) out of the host's control, and that's why it's said to be semi-autonomous.

One link layer protocol of particular importance is Ethernet. Ethernet is a multiple access protocol, meaning that it's used when many hosts are competing for a shared resource. Such situations occur on a local area network, or LAN. In Ethernet, if two packets are transmitted by different hosts at essentially the same time, they can collide, in which case both packets are corrupted. The packets must then be resent. The challenge is to efficiently handle collisions in a distributed environment. There are many possible ways to deal with a shared media, but Ethernet is by far the most popular method. In any respectable networking course, a significant amount of time is devoted to Ethernet, but we won't go into the details here.

While IP addresses are used at the network layer, the link layer has its own addressing scheme. We'll refer to link layer addresses as MAC addresses, but they are also known as LAN addresses, physical addresses, etc. MAC addresses are 48 bits, and they're globally unique. The MAC address is embedded in the NIC, and, unlike an IP address, it cannot change (unless a new NIC is installed). MAC addresses are used to forward packets at the link layer.

Why do we need both IP addresses and MAC addresses? An analogy is

often made to home addresses and social security numbers. A home address is like an IP address, since it can change. On the other hand, even if you move, your social security number stays the same, which makes it analogous to a MAC address. However, this doesn't really answer the question. In fact, it would be conceivable to do away with MAC addresses, but it is somewhat more efficient to use these two forms of addressing. Fundamentally, the dual addressing is necessary due to layering, which requires that the link layer should work with any network layer addressing scheme. In fact, some network layer protocols (such as IPX) do not use IP addresses and the link layer requires no modification to work with such protocols. The bottom line is that a strict adherence to layering requires that we have two distinct addressing schemes.

There are many interesting and significant link layer protocols. We've mentioned Ethernet and we'll mention just one more, namely, the Address Resolution Protocol, or ARP. The primary purpose of ARP is to find the MAC address that corresponds to a given IP address for hosts on the same LAN. Each node has its own ARP table, which contains the mapping between IP addresses and MAC addresses. This ARP table—which is also known as an ARP cache—is generated automatically. The entries expire after a period of time (typically, 20 minutes) so they must be refreshed periodically. Believe it or not, ARP is the protocol used to determine ARP table entries.

How does ARP work? When a node doesn't know a particular IP-to-MAC mapping, it broadcasts an ARP request message to every node on the LAN. The appropriate node on the LAN (i.e., the node with the given IP address) responds with an ARP reply. The requesting node can then fill in the corresponding entry in its ARP cache.

ARP is a stateless protocol, and as such, a node does not maintain a record of ARP requests that it has sent. As a consequence, a node will accept any ARP reply that it receives, even if it made no corresponding ARP request. This opens the door to an attack by a malicious host on the LAN. This attack, known as *ARP cache poisoning*, is illustrated in Figure A-6. In this example, the host with MAC address `CC-CC-CC-CC-CC-CC` has sent a bogus ARP reply to both of the other hosts, and they have updated their ARP caches accordingly. As a result, whenever `AA-AA-AA-AA-AA-AA` and `BB-BB-BB-BB-BB-BB` send packets to each other, the packets will first pass through the hands of the evil host `CC-CC-CC-CC-CC-CC`, who can alter the messages, delete the messages, or simply pass them along unchanged. This type of attack is known as a man-in-the-middle, or MiM, regardless of the gender of the attacker.

Recall that TCP provides an example of a stateful protocol that is subject to attack. ARP, on the other hand, is an example of a vulnerable stateless protocol. So stateless and stateful protocols both have the potential for security vulnerabilities.

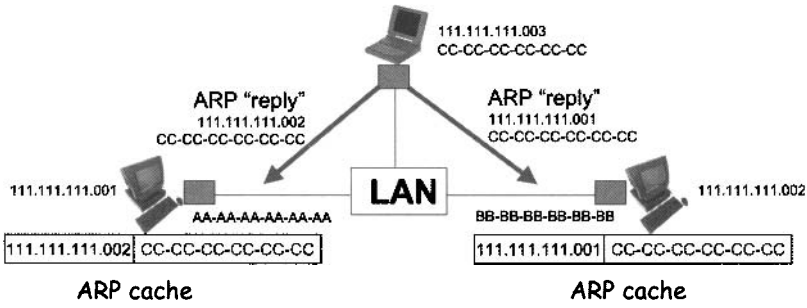

Figure A-6: ARP Cache Poisoning

A-1.7 Conclusions

In this section, we've barely scratched the surface of the vast topic that is networking. Tanenbaum [298] presents a good introduction to a wide range of networking topics, and his book is well suited to independent study. Another good introductory textbook on networking is Kurose and Ross [177]. A more detailed discussion of networking protocols can be found in [113]. If more details are needed than what is available in [113], consult the appropriate RFCs.