Network Security

Douglas W. Jacobson

Contents

Preface xiii

Acknowledgments xix

The Author xxi

Part I Introduction to Network Concepts and Threats 1

1 Network Architecture 3
1.1 Layered Network Architecture 3
1.2 Overview of a Protocol 12
1.3 Layered Network Model 15
Homework Problems and Lab Experiments 20
References 21

2 Network Protocols 23
2.1 Protocol Specifications 23
2.2 Addresses 29
2.3 Headers 35
Homework Problems and Lab Experiments 37
References 37

3 The Internet 39
3.1 Addressing 41
3.1.1 Address Spoofing 45
3.1.2 IP Addresses 46
3.1.3 Host Name to IP Address Mapping 47
3.2 Client-Server Model 49
3.3 Routing 54
Homework Problems and Lab Experiments 57
References 59

4 Taxonomy of Network-Based Vulnerabilities 61
4.1 Network Security Threat Model 61
4.2 The Taxonomy 69
4.2.1 Header-Based Vulnerabilities and Attacks 69
4.2.2 Protocol-Based Vulnerabilities and Attacks 70
4.2.3 Authentication-Based Vulnerabilities and Attacks 73
4.2.4 Traffic-Based Vulnerabilities and Attacks 75
4.3 Applying the Taxonomy 76
Homework Problems and Lab Experiments 78
References 79

Part II Lower-Layer Security 83

5 Physical Network Layer Overview 85
5.1 Common Attack Methods 87
5.1.1 Hardware Address Spoofing 87
5.1.2 Network Sniffing 89
5.1.3 Physical Attacks 90
5.2 Wired Network Protocols 92
5.2.1 Ethernet Protocol 92
5.2.2 Header-Based Attacks 101
5.2.3 Protocol-Based Attacks 101
5.2.4 Authentication-Based Attacks 102
5.2.5 Traffic-Based Attacks 104
5.3 Wireless Network Protocols 106
5.3.1 Header-Based Attacks 114
5.3.2 Protocol-Based Attacks 114
5.3.3 Authentication-Based Attacks 116
5.3.4 Traffic-Based Attacks 119
5.4 Common Countermeasures 124
5.4.1 Virtual Local Area Networks (VLANs) 124
5.4.2 Network Access Control (NAC) 126
5.5 General Comments 128
Homework Problems and Lab Experiments 129
References 131

6 Network Layer Protocols 135
6.1 IP Version 4 Protocol 137
6.1.1 IP Addressing 138
6.1.2 Routing 143
6.1.3 Packet Format 149
6.1.4 Address Resolution Protocol (ARP) 153
6.1.5 Internet Control Messaging Protocol (ICMP) 156
6.1.5.1 ICMP Echo Request (TYPE = 8) and Reply (TYPE = 0) 157
6.1.5.2 ICMP Timestamp Request (TYPE = 13) and Reply (TYPE = 14) 158
6.1.5.3 ICMP Destination Unreachable (TYPE = 0) 158
6.1.5.4 ICMP Time Exceeded (TYPE = 11) 159
6.1.5.5 ICMP Redirection (TYPE = 5) 159
6.1.6 Putting It All Together 159
6.1.6.1 Scenario 1 (H1 to H2) 160
6.1.6.2 Scenario 2 (H1 to H3) 162
6.1.6.3 Scenario 3 (H1 to H4) 164
6.1.6.4 Scenario 4 (H1 to H5) 166
6.1.6.5 Scenario 5 (H1 to No Host on Network 1) 168
6.1.6.6 Scenario 6 (H1 to No Host on Network 2) 170
6.1.7 Header-Based Attacks 172
6.1.8 Protocol-Based Attacks 173
6.1.9 Authentication-Based Attacks 174
6.1.10 Traffic-Based Attacks 177
6.2 BOOTP and DHCP 181
6.2.1 BOOTP Protocol 182
6.2.2 DHCP Protocol 185
6.2.3 Header-Based Attacks 186
6.2.4 Protocol-Based Attacks 186
6.2.5 Authentication-Based Attacks 189
6.2.6 Traffic-Based Attacks 190
6.3 IP Version 6 Protocol 190
6.3.1 Packet Format 191
6.3.2 ICMP Version 6 Protocol 194

6.4 Common IP Layer Countermeasures 195
6.4.1 IP Filtering 195
6.4.2 Network Address Translation (NAT) 196
6.4.3 Virtual Private Network (VPN) 203
6.4.4 IPSEC 206
Homework Problems and Lab Experiments 208
References 215

7 Transport Layer Protocols 221
7.1 Transmission Control Protocol (TCP) 221
7.1.1 Multiplexing 221
7.1.2 Connection Management 223
7.1.3 Data Transfer 223
7.1.4 Special Services 224
7.1.5 Error Reporting 225
7.1.6 TCP Protocol 225
7.1.7 TCP Packet Format 228
7.1.8 Header-Based Attacks 229
7.1.9 Protocol-Based Attacks 230
7.1.10 Authentication-Based Attacks 237
7.1.11 Traffic-Based Attacks 237
7.2 User Datagram Protocol (UDP) 238
7.2.1 Packet Format 239
7.2.2 Header- and Protocol-Based Attacks 239
7.2.3 Authentication-Based Attacks 239
7.2.4 Traffic-Based Attacks 239
7.3 Domain Name Service (DNS) 239
7.3.1 DNS Protocol 242
7.3.2 DNS Packet Format 245
7.3.3 Header-Based Attacks 248
7.3.4 Protocol-Based Attacks 248
7.3.5 Authentication-Based Attacks 248
7.3.6 Traffic-Based Attacks 250
7.4 Common Countermeasures 251
7.4.1 Transport Layer Security (TLS) 251
Homework Problems and Lab Experiments 253
References 254

Part III Application Layer Security 259

8 Application Layer Overview 261
8.1 Sockets 263
8.2 Common Attack Methods 266
8.2.1 Header-Based Attacks 266
8.2.2 Protocol-Based Attacks 267
8.2.3 Authentication-Based Attacks 267
8.2.4 Traffic-Based Attacks 268
Homework Problems and Lab Experiments 268
References 270

9 Email 271
9.1 Simple Mail Transfer Protocol 274
9.1.1 Vulnerabilities, Attacks, and Countermeasures 278
9.1.1.1 Header-Based Attacks 278
9.1.1.2 Protocol-Based Attacks 278
9.1.1.3 Authentication-Based Attacks 278
9.1.1.4 Traffic-Based Attacks 282
9.1.1.5 General Countermeasures 282
9.2 POP and IMAP 283
9.2.1 Vulnerabilities, Attacks, and Countermeasures 288
9.2.1.1 Header- and Protocol-Based Attacks 288
9.2.1.2 Authentication-Based Attacks 288
9.2.1.3 Traffic-Based Attacks 290
9.3 MIME 290
9.3.1 Vulnerabilities, Attacks, and Countermeasures 297
9.3.1.1 Header-Based Attacks 298
9.3.1.2 Protocol-Based Attacks 298
9.3.1.3 Authentication-Based Attacks 299
9.3.1.4 Traffic-Based Attacks 299
9.4 General Email Countermeasures 300
9.4.1 Encryption and Authentication 300
9.4.2 Email Filtering 304
9.4.3 Content Filtering 308
9.4.4 Email Forensics 309
Homework Problems and Lab Experiments 314
References 317

10 Web Security 321
10.1 Hypertext Transfer Protocol (HTTP) 324
10.1.1 Command Message 324
10.1.2 Response Message 326
10.1.3 HTTP Headers 326
10.1.4 Vulnerabilities, Attacks, and Countermeasures 333
10.1.4.1 Header-Based Attacks 333
10.1.4.2 Protocol-Based Attacks 334
10.1.4.3 Authentication-Based Attacks 334
10.1.4.4 Traffic-Based Attacks 336
10.2 Hypertext Markup Language (HTML) 340
10.2.1 Vulnerabilities, Attacks, and Countermeasures 343
10.2.1.1 Header-Based Attacks 343
10.2.1.2 Protocol-Based Attacks 344
10.2.1.3 Authentication-Based Attacks 344
10.2.1.4 Traffic-Based Attacks 344
10.3 Server-Side Security 345
10.3.1 Vulnerabilities, Attacks, and Countermeasures 347
10.3.1.1 Header-Based Attacks 347
10.3.1.2 Protocol-Based Attacks 348
10.3.1.3 Authentication-Based Attacks 348
10.3.1.4 Traffic-Based Attacks 348
10.4 Client-Side Security 349
10.4.1 Vulnerabilities, Attacks, and Countermeasures 351
10.4.1.1 Header- and Protocol-Based Attacks 351
10.4.1.2 Authentication-Based Attacks 351
10.4.1.3 Traffic-Based Attacks 352
10.5 General Web Countermeasures 352
10.5.1 URL Filtering 353
10.5.2 Content Filtering 356
Homework Problems and Lab Experiments 359
References 361

11 Remote Access Security 367
11.1 Terminal-Based Remote Access (TELNET, rlogin, and X-Windows) 368
11.1.1 TELNET 368
11.1.2 rlogin 372

11.1.3 X-Windows 376
11.1.4 Vulnerabilities, Attacks, and Countermeasures 378
11.1.4.1 Header-Based Attacks 379
11.1.4.2 Protocol-Based Attacks 379
11.1.4.3 Authentication-Based Attacks 379
11.1.4.4 Traffic-Based Attacks 381
11.2 File Transfer Protocols 382
11.2.1 File Transfer Protocol (FTP) 382
11.2.2 Trivial FTP 389
11.2.3 RCP 390
11.2.4 Vulnerabilities, Attacks, and Countermeasures 391
11.2.4.1 Header-Based Attacks 391
11.2.4.2 Protocol-Based Attacks 391
11.2.4.3 Authentication-Based Attacks 392
11.2.4.4 Traffic-Based Attacks 393
11.3 Peer-to-Peer Networks 394
11.3.1 Centralized Peer to Peer 396
11.3.2 KaZaA 399
11.3.3 Decentralized Peer to Peer 400
11.3.3.1 Limewire, Bearshare, and Gnutella 401
11.3.4 Vulnerabilities, Attacks, and Countermeasures 403
11.3.4.1 Header- and Protocol-Based Attacks 403
11.3.4.2 Authentication-Based Attacks 403
11.3.4.3 Traffic-Based Attacks 404
11.3.4.4 Peer-to-Peer Countermeasures 404
11.4 General Countermeasures 406
11.4.1 Encrypted Remote Access 406
11.4.2 SSH 407
11.4.3 Remote Desktop 410
11.4.4 Secure File Transfer (SFTP, FTPS, HTTPS) 411
Homework Problems and Lab Experiments 412
References 415

Part IV Network-Based Mitigation 425

12 Common Network Security Devices 427
12.1 Network Firewalls 427
12.2 Network-Based Intrusion Detection and Prevention 433
12.3 Network-Based Data Loss Prevention 437

Homework Problems and Lab Experiments 439
References 440

Appendix A Cryptology 445

Appendix B Laboratory Configuration 455

Appendix C Homework Solutions 461

Index 473

Part III

Application Layer Security

Part III of the book will look at a cross section of common network applications and their vulnerabilities and possible attacks. We will also examine countermeasures for each application vulnerability and attack. From a networking viewpoint applications can be categorized into three categories: store and forward, bulk transfer, and interactive.

Store-and-forward applications are characterized by the user's data being sent as a single packaged message. Store-and-forward messages are often not time critical. Often the messages are stored on intermediate computers as they traverse the network. Email is a good example of a store-and-forward application.

Bulk transfer applications also move blocks of information for the user, but the data moves between the client and the server, and time is more critical. These applications are the most common. They include file transfer applications, web-based applications, and file sharing applications.

Interactive applications are time critical and the data is broken up into small chunks that are transferred between the client and server. Interactive applications often involve a user interacting with an application directly and expecting a quick response on a keystroke-by-keystroke basis. These applications can generate a large number of small packets on the network. Typical interactive applications include remote access, voice or video streams, and interactive voice.

We will examine at least one application of each type, and since there are so many different applications, by using the taxonomy we can group the vulnerabilities and attacks. Such a grouping should be transferable to other applications not discussed in the book. The first chapter of Part III will provide an overview of the application layer and its interface with the transport layer.

Chapter 8

Application Layer Overview

Interface between the TCP layer and application layer was designed to be simple [1, 2]. The application layer assumes the transport layer will provide a connection-oriented reliable end-to-end transmission of data. There are some applications that are designed to work on top of the connectionless transport service, where the data is transferred using an unreliable service.

Before we look at specific application protocols we will first look at the interface between the application layer and the transport layer in terms of services, functions provided by the transport layer, and the parameters required and provided by the transport layer. This will provide a framework for looking at classifying which vulnerabilities and attacks are common to a large set of applications and which are application specific.

The transport layer is the typical interface between the application and the network. The transport layer is part of the operating system, and therefore has access to system resources. The application layer needs to connect to the transport layer and then use its services to transfer data. The two common transport protocols are User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). UDP is basically a direct connection to the Internet Protocol (IP) layer and provides only an application multiplexing service. The interface between the application layer and UDP is packet based. The TCP layer provides a reliable end-to-end connection between two applications. The TCP layer also provides flow control and error control. One interesting aspect of the interface between the application layer and the TCP layer is the way data is passed between them. TCP provides what is called a stream-oriented service, where the data is not presented as packets, but as a stream of data. TCP then takes the data and puts it in packets to pass to the IP layer. When TCP receives a packet from the IP layer, it places the data in a receive buffer, waiting for the application layer to read the data. An interesting side effect of the stream service is that the application layers need to parse the data to extract the application protocol, as shown in Figure 8.1.

As we see in Figure 8.1, user A wants to send a text message to user B. The application needs to add a header to the message as part of the application

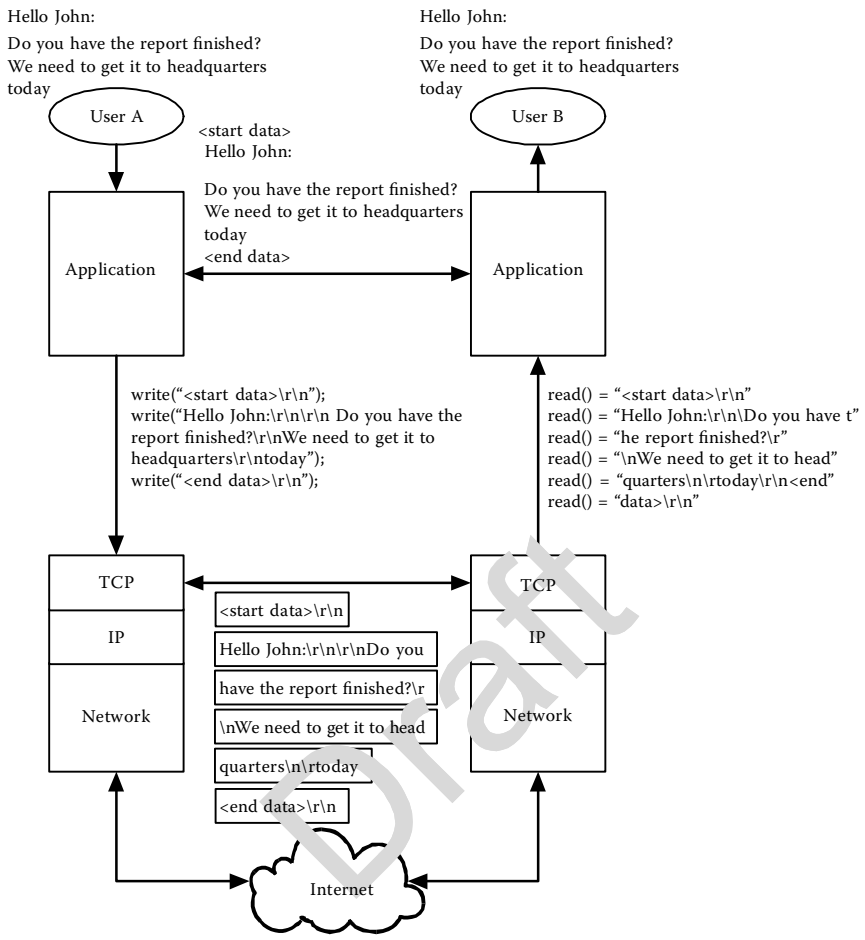


Figure 8.1: Stream service.

protocol. The header is a freeform header, which consists of <start data> and <end data>. The headers and the message are written to the TCP layer as a stream of data. Figure 8.1 shows the application sending the data that has the application header added to the TCP layer. In this example we assume that the application has already opened a connection to the corresponding application and has established a service connection with the TCP layer. As shown in Figure 8.1, the application layer sends the application to the TCP using three write functions. Depending on how the application is implemented, the application data can be presented to the TCP layer a byte at a time or as multiple bytes, as shown. Figure 8.1 shows the

TCP layer breaking the application data into six packets that are passed to the IP and then physical network layers. (Figure 8.1 does not show the TCP, IP, or physical network headers.)

The TCP layer decides when it has received enough data from the application to create a packet. This is based on the amount of data received and the time since the last data element was received from the application. The application can request that the TCP layer take all of the data it has so far and push it into a packet. This is helpful in application protocols that are interactive, like TELNET, Secure Shell (SSH), or chat applications. Figure 8.1 shows the receiving TCP layer presenting the data to the application via an application read request. TCP will provide the data it has assembled in its buffer when requested by the application layer. This can lead to multiple read requests per message. Figure 8.1 shows six read requests to get all of the data. The application needs to utilize its protocol to tell when the entire message has been received. In this case it looks for the string \r\n<end data>\r\n to tell when the message is finished. The \r and \n are the carriage return and linefeed characters that are generated when a user presses the enter key. In some cases the application protocol will use a length field to indicate the amount of data in the message, and in other cases it will use strings or other data markers. As mentioned above, TCP does support a method to allow the applications to push the data into a packet. This data push method will also cause the receiving TCP layer to push that data to the application during the read request. Because of the way the stream interface works, many of the application protocols are written almost like a conversation.

8.1 Sockets

Before we look at application protocols we need to understand how an application interfaces with the transport layer and how the application port numbering scheme works. As we saw in Figure 8.1, the application layer is provided with a stream service to send data. Before the client application can send data to the server application, it must first establish a connection with the operating system. After the connection has been made with the transport layer, the client application requests a connection with the server application using the TCP connection establishment protocol. The exact details of the programming interface between the application layer and the TCP layer are dependent on the operating system and the programming environment. We will look at a typical programming interface

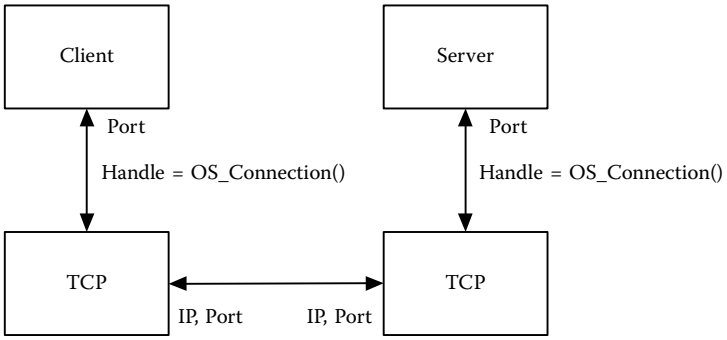


Figure 8.2: Application-TCP interface.

between the two layers and provide an overview of how the interface works. This interface is often referred to as a socket-level interface [3–7]. Figure 8.2 shows a client and server application where each application is using its respective TCP layer to communicate. The client and server applications have a connection to the TCP layer using the operating system. The operating system helps provide the multiplexing service of the TCP layer. Each application using the TCP layer will have a separate connection to the operating system. A connection to the operating system does not mean there is a connection with another TCP layer or application. A server, for example, will create its connection to the operating system and then wait for a client connection. As shown in Figure 8.2, each application has a port number and an IP address that creates the 4-tuple used to uniquely identify each packet, as discussed in Chapter 3. Once the TCP connection has been established, the application can use the stream service to communicate.

To get a better understanding of the sequence of events involved in establishing a client-server connection, we can refer to Figure 8.3. As we see in the timing diagram, the server starts by opening a connection to the TCP layer using the operating system. Since the server typically picks the port where it listens for a connection, the server will tell the TCP layer it wishes to be bound to a given port number. After binding to a port, the server can wait for the connection from a client. The TCP layer will indicate there is a connection request from a client, and the application can then decide if it will accept the connection.

On the client side of the diagram, we see that the client will also open a connection to the TCP layer. Since the client often does not care what port it uses, it will let the TCP layer choose a port number. Once the client is ready to connect to a server, it will issue a connection request to the TCP layer. The client

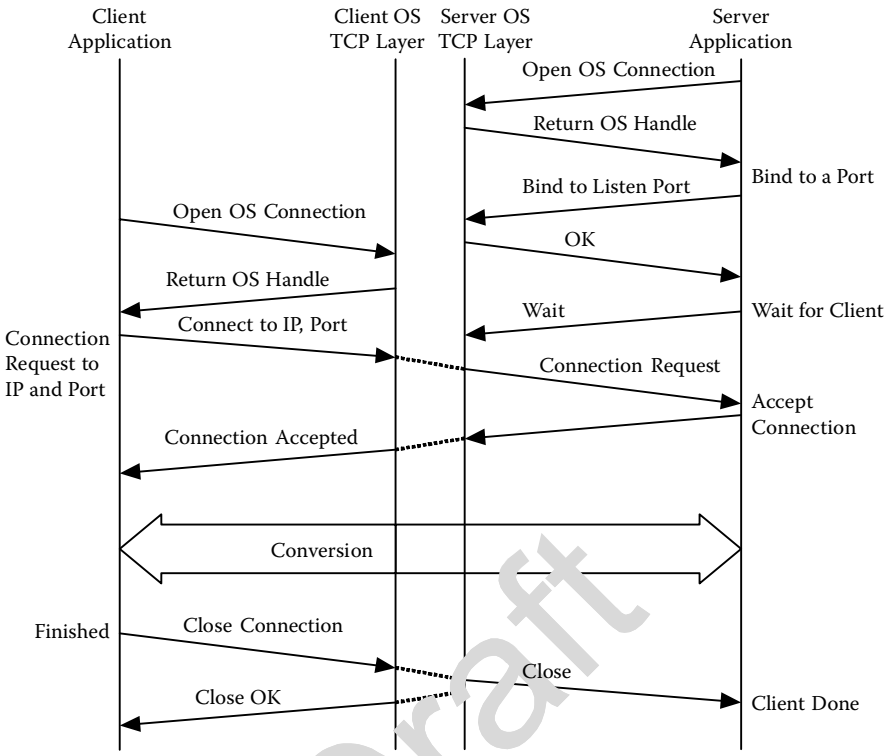


Figure 8.3: Socket protocol diagram.

will pass the IP address of the destination computer and the port number of the destination application to the connection request service call. The client TCP layer will establish a connection with the server TCP layer, and once the connection has been established, the client and server applications can communicate using the stream service. The diagram also shows the closing of the connection, which can be initiated by either application, although it is typically the client that closes the connection.

As we have seen from Figures 8.2 and 8.3, the interface between the application layer and the TCP layer is simple, which makes writing programs that open a connection and send data between each other easy. The complexity is in the application protocol and having the application programs exchange data that can be processed. Given this simple interface, there are not many attacks that can be played out against the interface; most attacks against the application target the protocol used by the application.

Definitions

Buffer overflow.
An attack method where an attacker sends too much data to a program, and when the program copies the data into an internal buffer, it overwrites its internal data.

Stream service.
Applications using TCP send data as a stream of bytes, not as packets.

TCP socket.
A connection between the application and the TCP layer. A pair of sockets is used by two applications to communicate with each other.

8.2 Common Attack Methods

This section will review the taxonomy presented in Chapter 4 as it relates to the application layer and common attack methods. One primary difference between attacks against the application layer and attacks against the lower layers is that the lower-layer attacks often affect all applications and can affect the operating system. An application-based attack can affect that application and sometimes allows the attacker to gain access to the computer system.

8.2.1 Header-Based Attacks

Header-based attacks are often used against the application layer. Since the application layer header is typically freeform, the implementation is often robust. The most common header attack is a buffer overflow. This attack happens when the data received is longer than the data expected. For example, if we refer to Figure 8.1, we see that the application header consisted of <start data>\r\n and <end data>. Let us also assume the designer of the software allocated a fifteen-character buffer to store the header. A buffer overflow occurs if an attacker sends a string that is longer than fifteen characters and the software designer did not check the length of the input. The extra characters are copied into memory and overwrite other variables in the program. Depending on how the other variables are used, the application program would crash, and in some cases the attacker could cause its own program to be executed. This is how worms are propagated through the Internet. Figure 8.4 shows an example of a buffer overflow. As we see

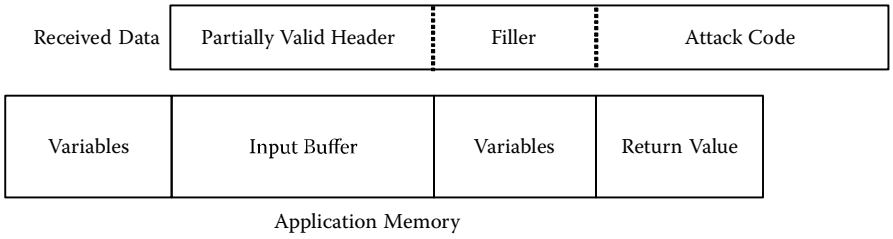


Figure 8.4: Buffer overflow.

in Figure 8.4, the received data contains the valid header followed by filler data, which is used to position the attack code into the right place in memory so that it can be executed. The length of the filler and the attack code are implementation dependent. This often means that a buffer overflow attack will behave differently on different systems. On one system the attack code may carry out the desired attack, and on another system it may just cause the application to fail, or it may not do anything.

8.2.2 Protocol-Based Attacks

Protocol-based attacks are very application specific. However, there are a few general observations that can be made about a protocol-based attack. Protocol-based attacks at the application layer often are part of an attempt to circumvent the authentication mechanism used by the application. One example would be using the protocol for authenticating an email user to try multiple usernames and passwords to try to gain access. Most protocol-based attacks only affect the application under attack. However, if the application provides remote access to the computer and the attacker gains access, he or she can affect the operation of the computer.

8.2.3 Authentication-Based Attacks

Authentication-based attacks are the most common type of attack against an application that requires authentication. These attacks can be part of a protocol-based or header-based attack. An authentication-based attack is application protocol specific. However, we can classify these attacks into two categories: direct attack and indirect attack.

A direct attack is where the attacker uses the authentication part of the protocol as a way to attack the application. An example of a direct attack is when the attacker responds to the application's request for a username and password.

An indirect attack is where the attacker uses one of the other attack categories (header, protocol, traffic) to circumvent authentication. As we saw in Chapter 4, authentication does not just involve a username and password. Applications might rely on IP addresses for authentication, and in many cases, applications perform no authentication of the user or of the other application. For the purpose of this book, we will focus on indirect authentication attacks. The direct attacks are more of a computer security issue, and the network is only used as a conduit for the attacker.

8.2.4 Traffic-Based Attacks

Traffic-based attacks against the application can cause the application to either quit or reduce its response time. Attacks designed to reduce the application's performance or deny access are called denial-of-service (DoS) attacks. There are some DoS attacks that are focused strictly at the application and operate by sending a large number of requests to the application over an open connection. Another type of DoS attack targets the other layers, but the goal is to deny access to the application.

The remainder of this part of the book will focus on several common network-based applications. We will start with an overview of each application protocol and then examine any security weaknesses in the protocol, followed by a set of possible solutions to the weaknesses. In some cases we will see that there are not any good solutions that can be implemented as part of the application, and that we will need to rely on other methods to mitigate the weaknesses.

Homework Problems and Lab Experiments

Homework Problems

1. In addition to the TCP socket there are local sockets. Research the uses of local sockets and comment on why local sockets are designed to work like TCP sockets.

2. Research buffer overflow attacks and develop a timeline of major attacks. Comment on why the buffer overflow vulnerability still exists.

3. Research the number of programming environments that support a socket interface.

4. Is there a limit on the number of TCP sockets that can be open at one time? What constraints create the limit?

Lab Experiments

1. Using the command "sockstat" (works on UNIX machines), get a listing of all sockets on a computer in the test lab. How can this command be used to help during a network attack?

2. Using the command "netstat -a," get a listing of all TCP sockets. Note that state of the sockets. Comment on the different states of the TCP sockets and how that relates to the state of the application.

3. Use tcpdump or wireshark to capture a web session between a computer in the test lab and another computer on the Internet. Look at the data in the packets to see how the application data is divided into packets.

Programming Problems

1. Download the file sock.tar from ftp://www.dougj.net. There are four source files in the tar file that are example TCP and UDP client and server programs. There will be programming problems in some of the following chapters that will expand on the program developed in this problem. Extract the files into a directory and type "make" to create the four programs. Note: There is a C and UNIX tutorial located on a web site described in Appendix B. There is a description of the programs in sock.tar and a brief tutorial on socket programming located on the web site. Perform the following:

    a. Run the programs tcp_server and tcp_client to see how they work. Modify tcp_server.c to accept and print data to screen indefinitely. Modify the tcp_client program to accept the destination port number as a parameter (flag of -p number). Use TELNET to connect to your tcp_server program, and send it data and comment on what you observe.

    b. Run the programs udp_server and udp_client to see how they work. Modify udp_server.c to accept and print data to screen indefinitely. Also modify the code to print out the IP address of the application that sent the packet. Modify udp_client to allow the user to input a string that will be sent as the packet. Use udp_client from multiple machines to test your code.

c. Modify the program tcp_client to accept a file as a parameter (flag -f filename) and to send the file to the tcp_server application.

References

[1] Forouzan, B. A., and S. C. Fegan. 1999. *TCP/IP protocol suite*. New York: McGraw-Hill Higher Education.

[2] Comer, D. E. 1995. *Internetworking with TCPIP*. Vol. 1. *Principles, protocols and architecture*. Englewood Cliffs, NJ: Prentice Hall.

[3] Comer, D. E., and D. L. Stevens. 1996. *Internetworking with TCP/IP*. Vol. iii. *Client-server programming and applications BSD socket version*. Upper Saddle River, NJ: Prentice-Hall.

[4] Stevens, W. R., and T. Narten. 1990. UNIX network programming. *ACM SIGCOMM Computer Communication Review* 20:8–9.

[5] Toll, W. E. 1995. Socket programming in the data communications laboratory. In *Proceedings of the Twenty-Sixth SIGCSE Technical Symposium on Computer Science Education*, Nashville, TN, 39–43.

[6] Schmidt, D. C., and S. D. Huston. 2001. C++ *network programming*. Reading, MA: Addison-Wesley Professional.

[7] Stevens, W. R., B. Fenner, and A. M. Rudoff. 2004. *UNIX network programming: The sockets networking API*. Reading, MA: Addison-Wesley Professional.

Chapter 9

Email

Email is one of the earliest network applications and one of the first to gain widespread use on the Internet [1–4]. The early email systems were proprietary and did not interoperate with other email systems. They used simple programs to create, send, and read short text messages and not support attachments. An email was sent through a series of servers that stored and forwarded the email as it traversed the network. The email server was used to send the outgoing email to the next server and received and stored the inbound email destined for the users. The users logged into the same computer that ran the email server. The demands on the functions and services offered by the email system grew over time, and today several protocols are involved to create the modern email system. As the email protocols have evolved, so have the vulnerabilities and attacks against email. Figure 9.1 shows the protocols involved to create, send, receive, and read email [5–7].

As shown in Figure 9.1, there are message transfer agents (MTAs), which are mail servers used to store and forward email. They communicate with a protocol called Simple Mail Transfer Protocol (SMTP). SMTP is an application protocol designed to use Transmission Control Protocol (TCP) connections to transfer mail from one server to another. Email can be stored on intermediate servers and then passed on to other servers. Each time the email is transferred from one server to another the SMTP protocol exchange is used. What the destination MTA does with the email once it is received is not part of SMTP protocol and is often implementation specific. The MTA also maintains its own file storage system for inbound mail that has not been picked up by the user and for outbound mail that is waiting for delivery. The MTAs do not authenticate each other and will allow any computer to connect and send email. Some basic authentication can be done to help increase security of the email system.

Another part of the mail system is the user agent (UA). The UA is the application that interfaces with the user and allows the user to create, read, send, and manage his or her email messages. As we see in Figure 9.1, there are two types of user agents (local and remote). Just like in the early email systems, the user agent can be on the same computer as the email server, as in MTA1 in Figure 9.1. In this

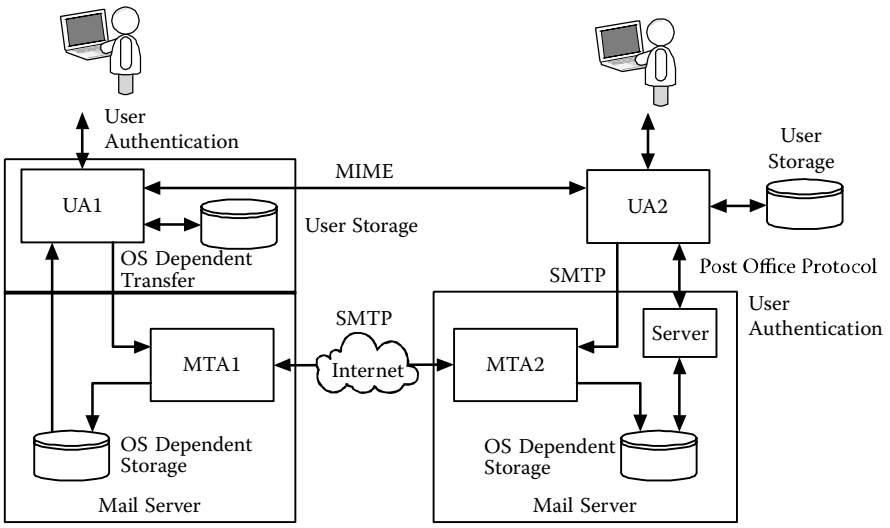


Figure 9.1: Email system.

case, the interaction between the user agent and the MTA is operating system and implementation dependent and often does not involve the network. The user agent also maintains storage to help the user manage his or her email messages.

The other type of UA is remote from the MTA, as in MTA2 and UA2 shown in Figure 9.1. In this case, the UA uses SMTP to send outbound email messages to the MTA, which will forward the messages on to the next MTA. One difference between the local UA and remote UA is that the remote UA uses a protocol to get the user's email off of the email server. There are several protocols that have been designed to accomplish this transfer. We will look at the Post Office Protocol (POP) and the Internet Message Access Protocol (IMAP). There are also web-based mail systems where the user retrieves his or her email using a web browser.

Another difference between the remote UA and the local UA is user authentication. With the local UA, the user is authenticated by the computer that is running the UA and is not authenticated by the MTA when the mail is read. With the remote UA the post office server authenticates the user before he is allowed to retrieve his email. In neither case is the outbound email authenticated; however, with the local UA the user still has been authenticated in order to gain access to the UA and MTA.

There is another protocol that is used by the UA when creating the email messages. The MTA does not care what is in the email message; it only needs to know the destination address. The user agents can use a protocol that will help

them display the message content. The most common format is Multipurpose Internet Mail Extensions (MIME) [8–12]. MIME is used to tell the user agents how the data is encoded and what type of data is in the email message. The email message may contain many different data formats, such as web-based data, pictures, text files, sound, and video. The user agents can display the content in a format that the user can use or see directly. For example, the user agent can display the pictures that were sent as part of the email message. The MIME protocol allows an attacker to send viruses, worms, and other malicious data directly to the user.

The electronic mail system was patterned after the postal mail system, and it helps to understand the email system if we compare it to the postal system. We can think of the MTA that accepts outbound email as the postal box on the corner, and the interaction between the MTAs as the postal system. Just as in the postal system, outbound email is not authenticated, which means anyone can send a letter with any return address by just placing it in the mailbox. Some MTAs do check to see if the return address is valid, but they have no way of telling if the user on the return address matches the actual user that sent the email.

The UAs can be equated to the user's mailbox. The UAs that are part of the MTA can be equated to home delivery of mail. The postal system delivers the mail to a home without authenticating the actual recipient. The assumption is that only people with access to the home have access to the mail. The UA that is remote from the MTA is similar to having your mail delivered to a post office box. In order to gain access to your mail you need to provide authentication (in the form of a key or combination).

The postal system does not open the letters as they are carried from the sender to the receiver, just as the MTAs do not open the contents of the email. There are a few exceptions where the MTA does open the message, such as spam filters and email virus scanners.

Before we examine the specific protocols, it would useful to look at the basic message format used by the mail system. An email message consists of a message header and the message body as shown in Figure 9.2.

The user sends a message that contains a picture and some text. The UA creates the message and adds a MIME header to indicate the email contains a picture and text. The UA can also add a header that contains information about the UA, the subject of the message, date and time the message was sent, and other information that might be useful to the recipient. The first MTA will take the message from the UA. Every time an MTA receives the message it will add a header to the front of the message. This header contains the date and time the message was received by

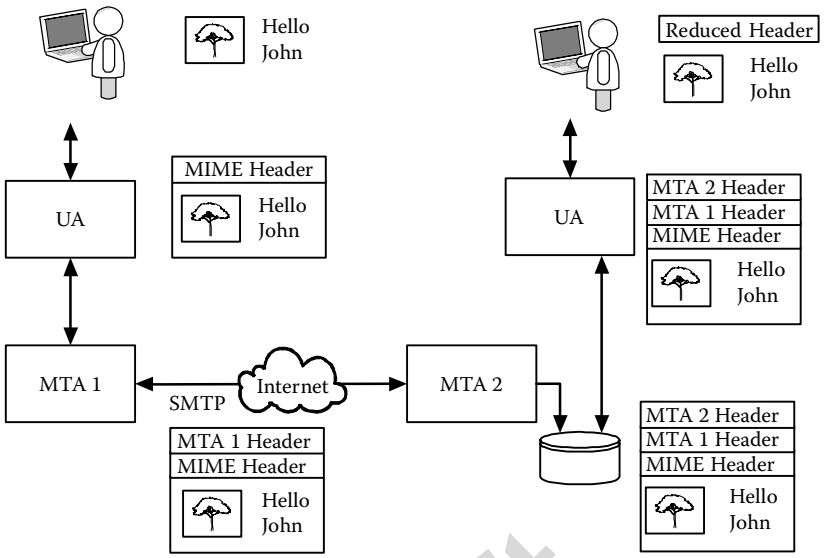


Figure 9.2: Email message format.

each MTA, the IP address and machine name of the sender, and the email address of the recipient. These headers can be useful in tracking the sender of an email message; however, most UAs will not display the headers in order to make email easier for the user. The user agent will present the picture and the text to the user along with a reduced header that typically consists of the sender's email address, the recipient's email address, and a time and date stamp.

The next sections of this chapter will examine the SMTP, POP/IMAP, and MIME protocols to see how they function, what vulnerabilities are common for each, and possible countermeasures to mitigate the vulnerabilities.

9.1 Simple Mail Transfer Protocol

The Simple Mail Transfer Protocol (SMTP) is designed to operate using the TCP stream service [13]. The email servers listen on well-known application port 25. The header format is freeform and consists of English words. The email message must be in 7-bit ASCII format. There are extended versions of the protocol that support binary data; however, 7-bit ASCII is the most common format. SMTP is

TABLE 9.1: Common SMTP Commands

Command	Action
HELO <domain>	Used by sending system to identify itself HELO machine.foo.bar
MAIL FROM: <path>	Identifies who the message is from MAIL FROM: john@issl.org
RCPT TO: <path>	Identifies who the message should be mailed to; there is a separate RCPT TO for each recipient. RCPT TO: mary
DATA	Indicates that the next transmission contains message text. The message is terminated by <cr><lf>.<cr><lf>.
RSET	Terminate current transaction
VRFY <user>	Returns the full name of the user specified (often not supported)
EXPN <alias>	Returns a list of mailboxes corresponding to the alias provided (often not supported)
NOOP	Returns a response code of "250 OK" and is used to test communication
QUIT	Ends the connection with the mail server
HELP	Shows a list of supported commands
EHLO <domain>	Request extended SMTP mode
AUTH	Authentication request
STARTTLS	Use Transport Layer Security

what is known as a command-and-response protocol. A command-and-response protocol is where one side issues commands (typically the client) and the other side sends a response message to each command. Table 9.1 shows the common commands supported by SMTP, and Table 9.2 shows the response messages. Depending on the implementation and configuration of the SMTP server, not all of these commands will be present or supported. Each command and response is terminated with a carriage return (<cr>) and line feed (<lf>).

SMTP is a command-and-response protocol, and just as the commands are in ASCII, the response codes are also in ASCII. Each response code consists of a three-digit ASCII number and a text field. The first digit of the response code indicates if the command worked or failed, the second digit specifies what type

TABLE 9.2: SMTP Response Codes

Code	Response Status
2XX	Positive completion reply—Indicates the command was successful and a new command can be issued
3XX	Positive intermediate reply—Indicates the command was successful, but the action is held up pending receipt of another command
4XX	Transient negative completion reply—Indicates the command was not accepted; however, the error is temporary
5XX	Permanent negative completion reply—Indicates the command was not accepted
Code	**Response Type**
X0X	Syntax error or unimplemented commands
X1X	Information—Reply to a request for information
X2X	Connections—Reply to a request for connection
X3X and X4X	Unspecified
X5X	Mail system—Indicates the status of the receiver

of code, and the third digit is used to indicate specific codes. The response code syntax is shown in Table 9.2. Some of the most common response codes are shown in Table 9.3.

Figure 9.3 shows a typical SMTP exchange between two MTAs used to send a message to a single user.

TABLE 9.3: Common SMTP Response Codes

Code	Responses
214	Help message
220	Service ready
250	Requested action completed
354	Start mail input
450	Mailbox busy
452	Requested action failed, insufficient system storage
500	Syntax error—command unrecognized
501	Syntax error in arguments
502	Command not implemented
550	Mailbox not found

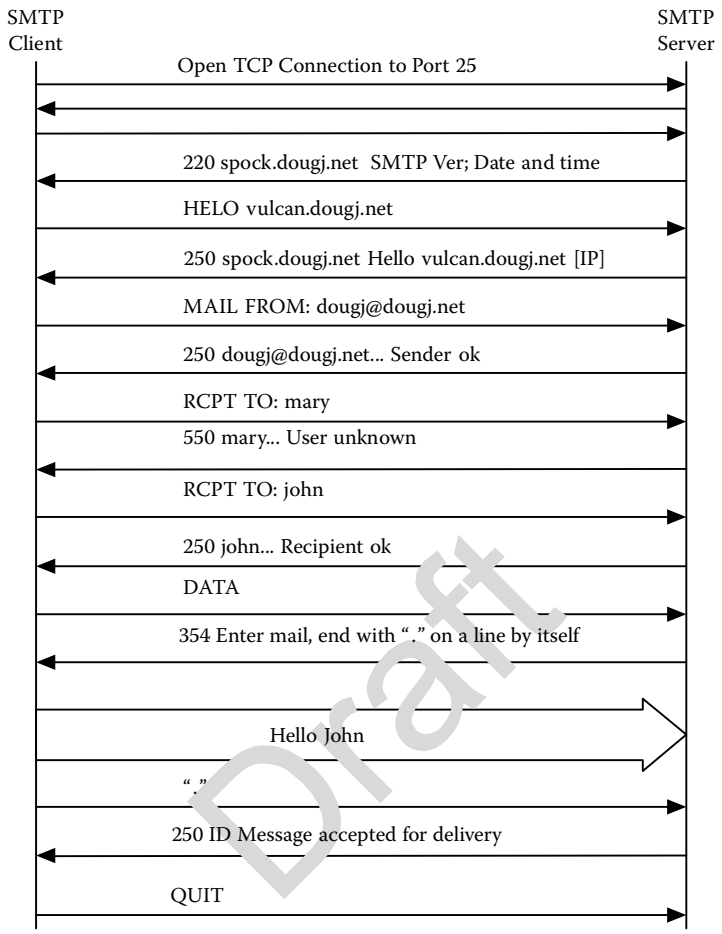


Figure 9.3: SMTP message exchange.

The client MTA opens a TCP connection with the server MTA. The server responds with a 220 response code. The text that follows the response code is representative of the response messages, but will vary depending on the implementation. The client then introduces itself using the HELO command and waits for a response code. The client tells the server where the email is coming from and who the email should be delivered to. It is up to the sender to tell the receiving MTA where the email came from. Notice that the command "RCPT TO mary" failed because mary is not a user the server knows about. Also note that the response to the HELO contains the IP address of the client, which was obtained from the TCP layer. Following the DATA command the client sends data until a

"." is on a line by itself. There is no message size indicator. The client can then either send another email message by sending another set of MAIL FROM: and RCPT TO: commands or can quit by sending the QUIT command.

9.1.1 Vulnerabilities, Attacks, and Countermeasures

Even though the SMTP is straightforward, there are several vulnerabilities. Two of the four categories of vulnerabilities and attacks, discussed in Part I, are common.

9.1.1.1 Header-Based Attacks

Header-based attacks are not very common since the headers are simple and any command or response that is invalid is ignored. The early versions of email servers were subject to buffer overflow attacks. The early implementations had a fixed buffer size for the SMTP commands and responses. There have been several well-known attacks that were able to exploit the fixed buffer size by sending commands that were too long. Today most email servers have been patched or designed to accept arbitrarily long input messages. They typically toss all input commands or responses that are over a certain length. However, there is attack code still available that targets SMTP servers using command line buffer overflow. Often a large percentage of attacks that are launched against a particular service or protocol have no chance of working. This makes defending a network even more difficult.

9.1.1.2 Protocol-Based Attacks

Protocol-based attacks are not common in command-and-response-based protocols since the timing and sequence of protocol messages are controlled and any violation is ignored.

9.1.1.3 Authentication-Based Attacks

The most common category of email attack is an authentication-based attack, which is due in large part to the lack of authentication in the SMTP protocol. The most common SMTP authentication attack involves using a fake sender either directly or through a process called relaying. This attack is known as email spoofing. As was shown in Figure 9.3, the client is responsible for telling the server the email address of the sender. There is no process or protocol to verify the sender of the email message. (Several have been proposed, but are not widely adopted.) The only countermeasure widely deployed is to examine the sender's domain to see if it is valid, which can be done using a domain name lookup

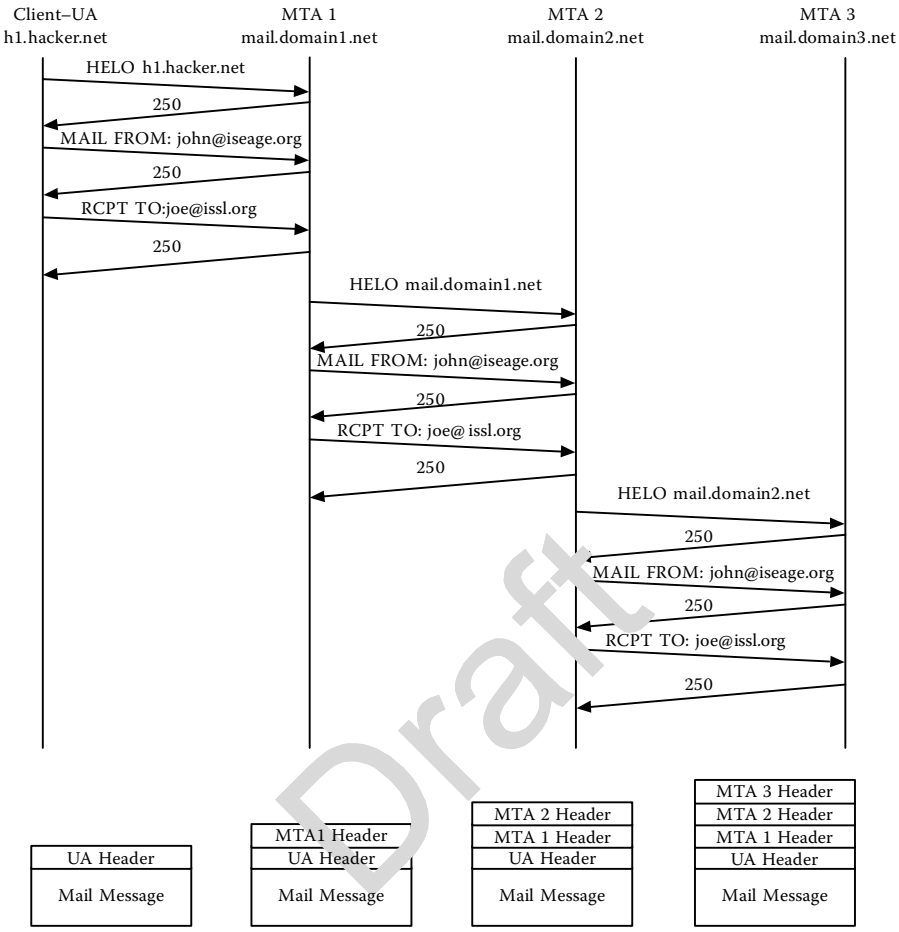


Figure 9.4: Sender and receiver address propagation.

protocol, Domain Name Service (DNS). So looking at Figure 9.4, for example, MTA 1 would check to see if "iseage.org" is a valid domain name [14]. There is no protocol in place to verify the user "john" as being a real user in the domain "iseage.org."

In addition, there is no attempt by MTA 1 to correlate the name of the client computer, h1.hacker.net, with the sender's reported email address, iseage.org. This is due to the fact that the email message may be forwarded through several MTAs before reaching the destination, and each MTA client communicates the "from" and "to" email addresses to the MTA server. In Figure 9.4 we see an email message propagating through several MTAs, each adding its own header and each using the original sender's and receiver's addresses.

Email address spoofing is used for spam email and other malicious email messages. There are several methods that can be used to send email with a spoofed address, including setting the return address on the UA. Spammers use custom software that interacts with the MTA using the SMTP protocol. Since the UA only shows a reduced header to the user, the user cannot easily tell that the message has a spoofed sender's address. The other issue with a spoofed sender's address is if the final recipient's address is invalid, then the mail message is bounced back to the spoofed sender's address. If the address the attacker uses as the return address actually exists, then this could cause excess email traffic to be sent to the spoofed address and could cause the system's disk storage to fill on the spoofed email server. This type of attack could also be considered a traffic-based attack.

Another feature of the email system allows a remote user to spoof the return address. This feature allows remote UAs connected to a single email server to have a return address that is the organization's address rather than the sender's computer address. Figure 9.5 shows a scenario where three user agents are connected to a single MTA for mail handling.

The UAs are running on three different hosts, each with a different machine name and IP address. The goal is to have a consistent view from the outside, so all email looks like it comes from the same main mail server and not from each individual host. For outbound email, the sender's address should be user@iseage.org even though he or she is on a machine that is not iseage.org. Inbound mail would be destined for user@iseage.org, and each user would authenticate with the MTA to pick up his or her mail using a post office protocol. In order to handle changing the sender's address, the MTA is set up to allow forwarding of email messages through a method called relaying. When a message is relayed the MTA forwards the message on to the MTA with the sender's address of user@iseage.org and the recipient's address of user@domain. In Figure 9.5, three users (Mary, John, and Jill) are sending a mail message to john@issl.org. Each UA has been configured to have a return address of user@iseage.org. When the UA sends the email to MTA 1, it uses the RCTP TO: address of john@issl.org. MTA 1 takes the mail and forwards it on to the next MTA for delivery. The detailed header will show that the email is from the UA on the host; however, the "from" address used in the header as the return address is user@iseage.org.

Another aspect of relaying is the ability for the sender to specify the route that the email will take as it traverses the network called source routing. This is used to send the email through a set of internal MTAs. It is not common to specify the source route across multiple remote MTAs since each MTA involved in the

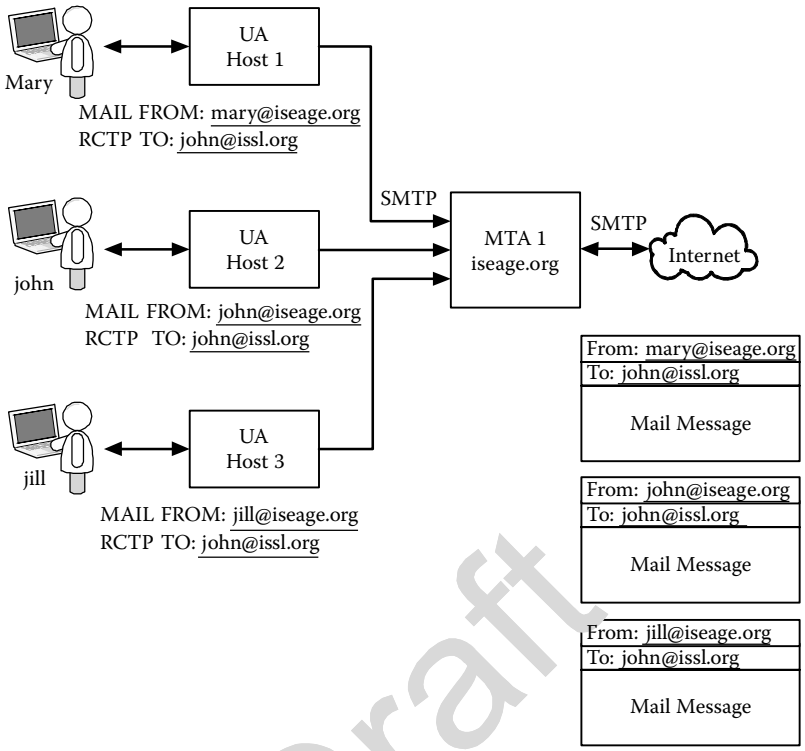


Figure 9.5: Email relay.

source route will need to accept relaying. From a security standpoint there is no real difference between email forwarding and source routing.

A problem with email relaying is that an attacker can use a relay to fake the return address and use the victim's mail server to send spam and malicious emails. There are public domain attack tools that search for an MTA that allows relaying. All an attacker needs to do is to open the connection to the MTA and send a RCPT TO: user@domain, and if the MTA responds with a 250, then it will accept relaying. Most MTAs that allow relaying restrict who can send a relayed message based on the IP address of the machine connected to the MTA server. This is not a perfect solution since IP addresses can be spoofed, and if a machine in the acceptable IP address range has been compromised, then it could be the source of the relayed email.

Another authentication-based attack is username probing. This attack is simple to carry out, but has minimal security implications. As we saw in Figure 9.3, the RCPT TO: command returns an error if the user is not valid. This could be

used to find usernames, or at least to verify usernames. If the attacker has no information about the target, using SMTP to guess usernames would be very time-consuming and would be logged. Most of the time email usernames are known, and therefore it would not gain the attacker much additional information by probing for usernames.

9.1.1.4 Traffic-Based Attacks

In addition to authentication-based attacks, traffic-based attacks can be played out against email systems. The most common attack is to flood the email server with large messages that consume the disk storage. As the size of disk storage increases, this attack is becoming less useful. In addition, many email systems place a quota on the size of an inbound mailbox, which causes this type of attack to only impact individual users and not the entire system. There have been some clever flooding attacks where you get an email server to respond to a spoofed return address. An attacker can send email from machine A to machine B with a return address of machine C. Machine B would respond back to machine C. Another common flooding attack is often accidental and happens when a user sends an email using a relay to a large list of users. If we remember from Figure 9.3, the sender can send the command RCPT TO: multiple times. This will cause the relaying MTA to make a copy of the message for each outbound destination. If the email message is large, the user could fill up the outbound message queue. However, the outbound queue usually empties as the MTA makes contact with the destination MTAs.

Another traffic-based attack is sniffing the SMTP traffic. Sniffing traffic from the Internet is difficult, but an attacker could sniff the traffic inside an organization if he or she has gained access to the organization's network. The SMTP protocol is not encrypted, so an attacker would be able to read the email messages. The next section describes a couple of encryption methods for the SMTP traffic.

9.1.1.5 General Countermeasures

In addition to the vulnerabilities and countermeasures described above, there are several additional authentication-based countermeasures that have been proposed to aid in email security. These are the STARTTLS and AUTH commands in the SMTP protocol. STARTTLS is used to negotiate the parameters for the Transport Layer Security protocol [15–18]. This protocol provides an authenticated and encrypted delivery of mail to an MTA. This does not provide end-to-end authentication and encryption of the message. Nor does it provide user-to-user authentication and encryption.

AUTH provides a mechanism for users connecting to an MTA to authenticate themselves for the purpose of sending outbound messages. Both STARTTLS and AUTH are typically used for remote access to the MTA for relaying. For example, a user that takes his laptop on the road and needs to send email as if he is part of the home network would use these protocols to prove to the MTA that it could relay messages. These protocols are not used to secure email messages between two end users, nor do they help in fighting spam. There is an ongoing debate about whether email delivery using SMTP should be an authenticated service as a way to reduce spam. Another idea is to add a charge for email, which would also require authentication. It is this author's belief that email delivery should remain unauthenticated and that email security should be handled by the user agents.

Definitions

Email relay.
Sending email messages with a new return address that matches the return address of the organization's email server. This allows multiple internal MTAs, and clients look like they are using just one MTA.

Email spoofing.
The process of creating an email message where the sending address is not the same as the address of the actual sender.

Message transfer agent (MTA).
The application that handled the reception and delivery of email messages. Its operation is similar to the postal system.

Simple Mail Transfer Protocol (SMTP).
The protocol used to transfer email messages between MTAs and from an email client to an MTA.

User agent (UA).
The name given to the client application that is used to create and read email messages.

9.2 POP and IMAP

As shown in Figure 9.1, the user has more than one way to access her inbound email. Figure 9.6 shows three methods for the user to access her email stored on the MTA. In one method (local user agent), the inbound mail messages are stored

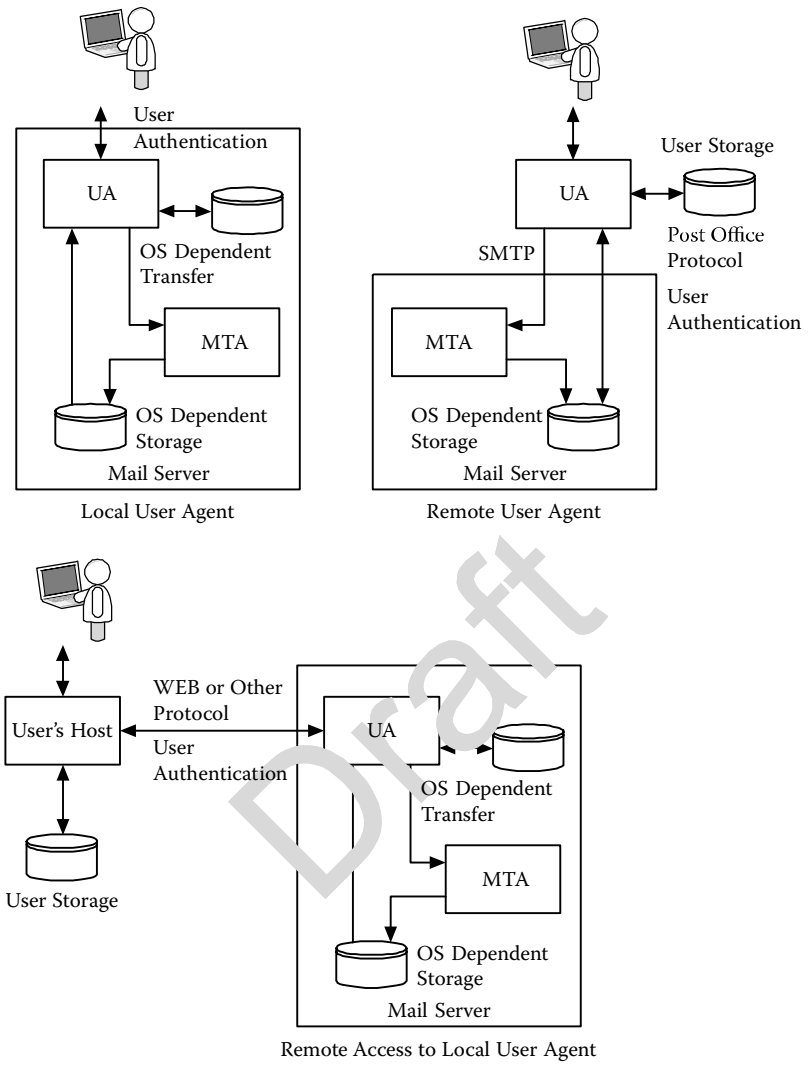


Figure 9.6: User agent access.

on the same system where the user has an account. The user authenticates herself with the server (typically by logging into the server) and then runs the user agent application, which has direct access to the user's inbound emails. This method does not rely on any networking other than the user may use network protocols to access the server.

In the second method (remote access to local user agent), the user remotely accesses the user agent running on the server. The most common implementation

of this method is web-based email. The security implications of this method fall under the discussion of web applications.

In the third method (remote user agent), the user agent is remote from the mail server and the user agent uses a network protocol to access and transfer the email messages to the remote user agent. There are two common protocols used to support access to email by a remote user agent. The first is Post Office Protocol (POP), and the second is Internet Message Access Protocol (IMAP) [19–21]. The basic functionality of each protocol is similar, and the security issues are common.

POP is used to transfer email from a mail server to a computer running a user agent. POP provides the ability to preview email and limited management of remote email. POP also provides user authentication before access to the email is granted. POP is patterned after the real post office where a user has a P.O. box with some type of user authentication. The P.O. box is designed to be intermediate storage of mail until the user retrieves the mail and takes it with her or throws it away while at the post office. The POP protocol works the same way. The user authenticates with the server, and then she can look through the email and decide what should be thrown away and what should be transferred to the user's computer. Like many protocols, POP has gone through several versions, and for the purpose of this text, we will focus on version 3, which is the most current.

Version 3 of POP (often called POP3) uses TCP to connect the client's user agent to the server. The POP3 server listens on port 110 and waits for a client to connect and for the user to authenticate. The POP3 protocol is a command-and-response protocol like SMTP. The POP3 commands and response codes are shown in Table 9.4.

There are two response codes: ERR indicates an error and OK indicates the command was successful. (Note that ERR starts with a − and OK starts with a +.)

Figure 9.7 shows the typical interaction between the client and server using the POP3 protocol. Note that the username and password are sent to the server with no encryption. If the username is incorrect, most POP3 servers will still prompt for a password, thus not letting an attacker know if he or she has guessed a valid username. How is this different than the way SMTP handled the RCPT TO: command? The major difference is that the email address for the user handled by the SMTP server may not be the username on the server, whereas the usernames in the POP3 protocol are valid usernames on the server. This means an attacker could use that username to log in to the server if he or she knew the password.

TABLE 9.4: Common POP3 Commands and Responses

Command	Action
USER name	User name for authentication
PASS string	Send the password for the user
STAT	Returns the number of messages
LIST [msg]	Returns the size of message or the size of all messages if none is specified
RETR msg	Send full message to client
DELE msg	Delete message from server
NOOP	No operation, returns OK status code
RSET	Reset deletion indicators
QUIT	Quit the session
TOP msg n	Returns the first n lines of message
UIDL msg	Returns a unique id string for the requested message; does not change during the session
Response Codes	Action
–ERR message	Error
+OK message	Command was successful

As shown in Figure 9.7, once the POP3 client has been authenticated, the server responds back with the number of email messages. The client can then retrieve any message and can also delete any message. Note the messages are not deleted until the session ends with the QUIT command. When the client retrieves a message, it uses the message number and is given the number of bytes in the message, and then the message is sent by the server. Also note that the message is terminated with a <cr><lf>.<cr><lf>.

The POP3 protocol is designed to facilitate the transfer of email messages from the server to the client, but if a user has multiple computers where she accesses her email, then there is problem keeping the email in sync, since the email will end up transferred to several different computers. Since POP is not really designed to keep the email on the server, another protocol was created: the Internet Message Access Protocol (IMAP).

There are a couple of differences between IMAP and POP. IMAP supports mailboxes on the server as a way to organize email. IMAP also supports moving messages between the client and server folders, searching the server folders, and managing the server folders. When the user stores her email on the server, she

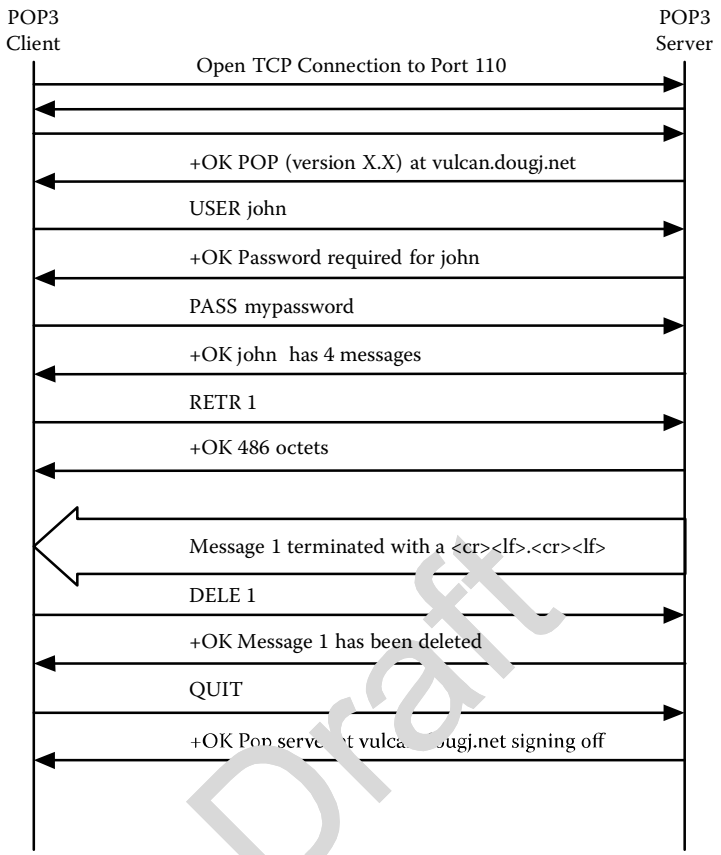


Figure 9.7: POP3 protocol diagram.

can easily move between multiple clients. Figure 9.8 shows how mailboxes are handled in a typical IMAP implementation. There are user mailboxes on the server that can be accessed by any user agent that knows the username and password associated with the mailboxes. The UA can use the IMAP protocol to create, manage, and delete remote mailboxes and to move email between the local and remote mailboxes. The user can create a number of mailboxes to help her organize her mail.

IMAP is a command-and-response protocol. A subset of the IMAP commands is shown in Table 9.5. IMAP is a more complicated protocol and provides better support for multiple remote user agents. From a security standpoint, IMAP and POP3 have the same security concerns, and therefore we will not explore the IMAP protocol in detail.

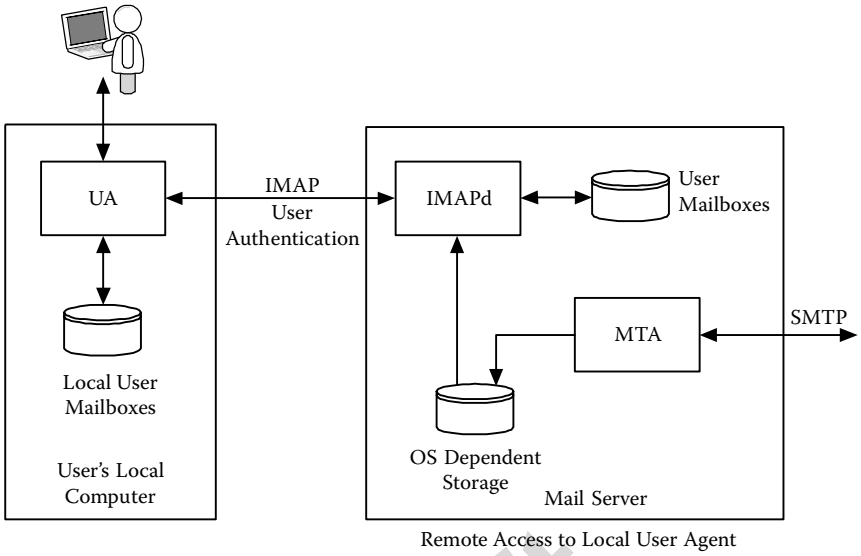


Figure 9.8: IMAP mailboxes.

9.2.1 Vulnerabilities, Attacks, and Countermeasures

9.2.1.1 Header- and Protocol-Based Attacks

POP and IMAP have not had many header and protocol vulnerabilities over the years. Both protocols are simple and the headers are freeform text.

9.2.1.2 Authentication-Based Attacks

There are numerous network applications that require a user to authenticate over the network with a username and password. This allows an attacker to send usernames and passwords in an attempt to break into the computer. POP will allow a user to try an unlimited number of login attempts. The code to carry out a password-guessing attack against a POP or IMAP would be easy to write, but the probability of the attack being successful is low. There are too many possible password combinations to guess every password, so attackers often rely on a dictionary of words. See Appendix A for more details on the strength of passwords. These types of direct password-guessing attacks can be successful against configuration vulnerabilities, where the default passwords have not been changed. With POP and IMAP, every login attempt is logged along with every email transaction. However, the log file becomes quite large and is often overlooked by system administrators. There are not any good countermeasures for

TABLE 9.5: Common IMAP Commands

Command	Action
CAPABILITY	Returns a list of capabilities the server supports
NOOP	No operation
LOGOUT	Ends the session
AUTHENTICATE type	Indicates the client wants to authenticate using the specified type
LOGIN name password	Log in using username and password
SELECT mailbox	Select the mailbox to be used (inbox is the inbound mail queue)
EXAMINE mailbox	Read-only version of SELECT
CREATE mailbox	Create a new mailbox on the server
DELETE mailbox	Delete the mailbox on the server
RENAME current new	Rename the current mailbox to a new name
CLOSE	Close the mailbox and all mail marked to be deleted will be deleted
EXPUNGE	Delete all messages marked for deletion
SEARCH criteria	Search the mailbox for messages that match the criteria
FETCH message items	Fetch the message items. (Note: The client can request the entire message or parts of the message or information about the message.)

remote user authentication. One method is to limit where a user can authenticate, from using the IP address of the user's computer, for example, only allowing POP or IMAP access from a certain IP address range. This can also be accomplished by not allowing the POP or IMAP protocols to cross the perimeter of the network through the use of firewalls or screening routers. These types of network-wide solutions are discussed in Part IV of the book. We find that most users want access to their email from wherever they are, which makes restricting user access to an IP address range or to the organizational network not practical.

Depending on the level of security desired, some organizations use VPN software on remote clients, which provides an encrypted and authenticated connection into the organization's network. With VPN software the organization can restrict the POP and IMAP protocols to only run inside their network. Another countermeasure is to not allow remote access to POP or IMAP, but allow remote email access through a web client, as shown in Figure 9.6. Web servers have their own

authentication system and support encrypted traffic. Some organizations have adopted web-based user agents for all email access.

9.2.1.3 Traffic-Based Attacks

POP and IMAP could be subject to traffic-based attacks, but the damage would be limited. An attacker has little to gain from trying to overwhelm a POP or IMAP server with traffic since a new server is started for every user connection.

A vulnerability that exists in most protocols is that the data is transferred in clear text, and therefore any attacker that can sniff the traffic can read the data. As we described in Chapter 3, the attacker needs to be connected to the network where the traffic is passing to be able to sniff the packets. This vulnerability has different effects, depending on the application. In the case of POP and IMAP, the biggest issue is that the username and password are transferred in clear text, and that an attacker could capture usernames and passwords. The general countermeasure is to use encryption technologies to ensure the data cannot be read. There are secure versions of both POP and IMAP that use public key encryption to exchange a symmetrical key. All traffic is then encrypted, including the username and password exchange. Appendix A provides an overview of encryption methods. While these secure versions do fix the clear text problem, they are not widely used. Most of the newer user agents support secure IMAP and POP that use Transport Layer Security (TLS). TLS was discussed as a countermeasure in Chapter 7.

Definitions

Internet Message Access Protocol (IMAP).
A protocol used by a user agent to transfer email from an MTA to the user computer. IMAP also allows for the creation and maintenance of mailboxes on the MTA.

Post Office Protocol (POP).
The protocol used by a user agent to transfer email from an MTA to the user's computer.

9.3 MIME

In this chapter we have looked at protocols to move email from one server to another and protocols used to transfer email from the server to the user agent. There is another protocol, Multipurpose Internet Mail Extensions (MIME), that

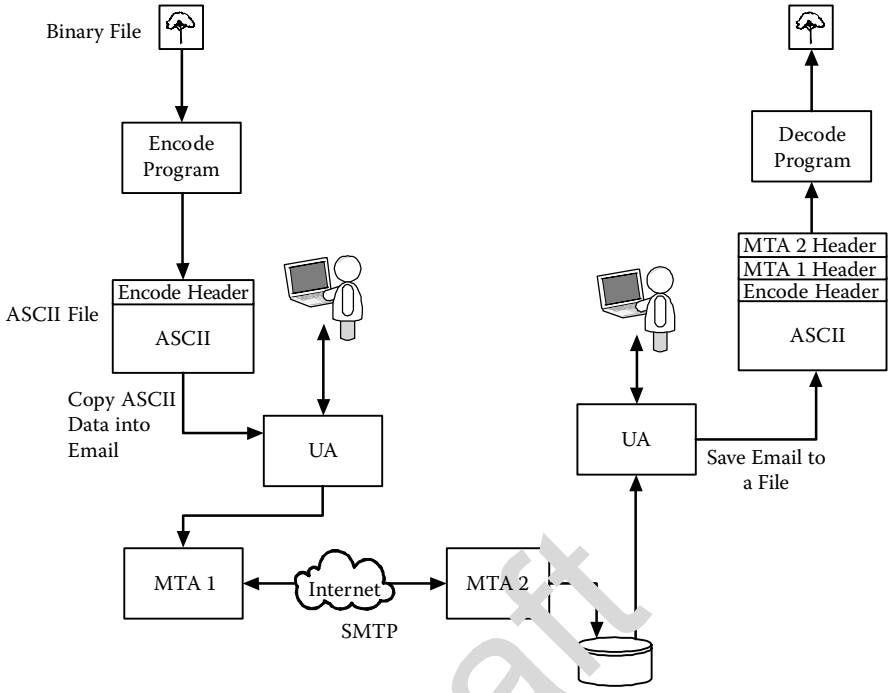


Figure 9.9: Email encode and decode methods.

is used to format the email messages themselves [8–12]. MIME is more of a message format than an actual protocol. From a network security standpoint, MIME is worth discussing since it is used to transport any data type and present it to the user. The MIME protocol has been used to transport viruses, worms, and other malicious code. MIME also enables inserting pictures and web links into email messages, which are used in spam and identity theft. In this section we will briefly discuss the MIME protocol and then look at how the protocol is used to carry out attacks.

SMTP, designed to carry 7-bit ASCII data, worked well in the early days of email. However, even in the early days of email, users wanted to send more than simple text. They created a simple encoding method that converted binary data into ASCII data. A user would first take the binary file and encode it, and then the user would email the file to another user. The receiving user would store the email message as a file and then strip away the headers and pass the message through the decode routine. This is shown in Figure 9.9. The encode and decode functions were separate from the user agent, and several different methods were created to

TABLE 9.6: MIME Headers

Header	Function
MIME-Version	Indicates a MIME message. The current version is 1.1.
Content-Type	Indicates the type of content contained in the message
Content-Transfer-Encoding	Indicates how the content is encoded
Content-Id	Optional identifier used for multiple messages
Content-Description	Optional description of the object that can be displayed by the user agent
Content-Disposition	Optional description of the method to use to display the object in receiving the user agent

enable users to transfer non-ASCII data. As you can imagine, this was not very user-friendly.

As the sophistication of the user agent programs increased, and with the advent of graphical user interfaces, there was a need to design a protocol that the user agents could use to exchange non-ASCII data. MIME was designed to support arbitrary data formats and allow for new data formats. MIME has a freeform header format, and unlike typical network protocols, there is no exchange of packets. The sending user agent assembles a MIME message and includes that as part of the message. The receiving user agent will interpret the message using the MIME headers to decode each part. The receiving user agent does not need to interact with the sending user agent to decode the messages. Another interesting aspect of MIME is that since the MIME encoded message is in ASCII, non-MIME user agents can receive the message. However, they are not able to display that data in the original format. MIME also supports messages that can be seen with both MIME and non-MIME user agents.

The MIME protocol has three required headers and three optional headers that are included in the message given to the message transfer agent by the user agent. In Figure 9.2 we saw that the message created by the sending user agent contained a MIME header, and that header was read by the receiving user agent. The MIME headers are shown in Table 9.6 and are described in the following paragraphs.

The first header indicates the message is in MIME format and is found once in the message. As shown in Figure 9.10, the other headers can appear multiple times in the message, typically once for each object in the message.

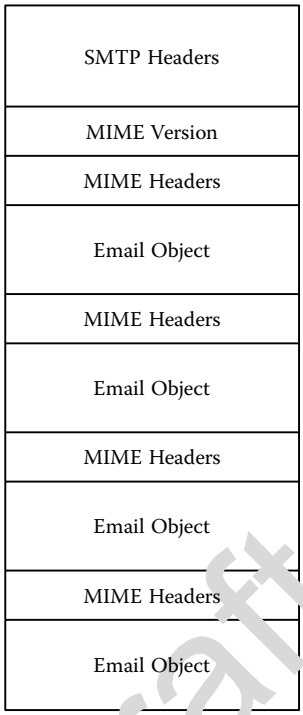


Figure 9.10: MIME headers.

The following list will look at each of the six headers and will give some examples of each header in use. We will then look at several different attacks that use MIME and how to mitigate them.

MIME-Version: The MIME-Version is used to indicate that the remainder of the message is in MIME format. The syntax is:

```
MIME-Version: 1.1
```

Content-Type: The Content-Type header is use to indicate what type of objects are contained in the message. The syntax of the header is:

```
Content-Type: type/subtype; parameters
```

The MIME protocol supports several object types, each with one or more subtypes. Table 9.7 shows many of object types supported by the MIME protocol. These types and some of the

TABLE 9.7: MIME Object Types

Type	Subtype	Description
Text	Plain	Unformatted text
	HTML	Text in HTML format
Multipart	Mixed	Multiple ordered objects
	Parallel	Multiple object, not ordered
	Digest	Multiple ordered RFC 822 objects
	Alternative	Alternate methods of representing the same object
Message	RFC 822	Encapsulated message
	Partial	Part of a larger message
	External—body	Object is a reference to an external message
Image	JPEG	JPEG image
	GIF	GIF image
Video	MPEG	MPEG movie
Audio	Basic	Audio object
Application	Postscript	Adobe Postscript object
	Octet-stream	8-bit binary object

subtypes are described below. It is not necessary to completely enumerate all of the MIME headers for us to understand the security implications and to develop methods to mitigate the vulnerabilities.

Content-Type: Text/Plain: Seven-bit ASCII text that is displayed by the receiving user agent in a format that it chooses. This is used by MIME-capable agents when the user sends a text message. A non-MIME user agent that receives the message will display the MIME headers along with the text message. Text/Hypertext Markup Language (HTML) is used to transfer HTML-formatted documents. This has become commonplace in email messages and allows users to create email text that is formatted. This also allows users to create email messages that look like web sites and can contain hyperlinks to web sites. This is also used by spammers and other attackers to get users to click on a link and either buy something or provide them with personal information.

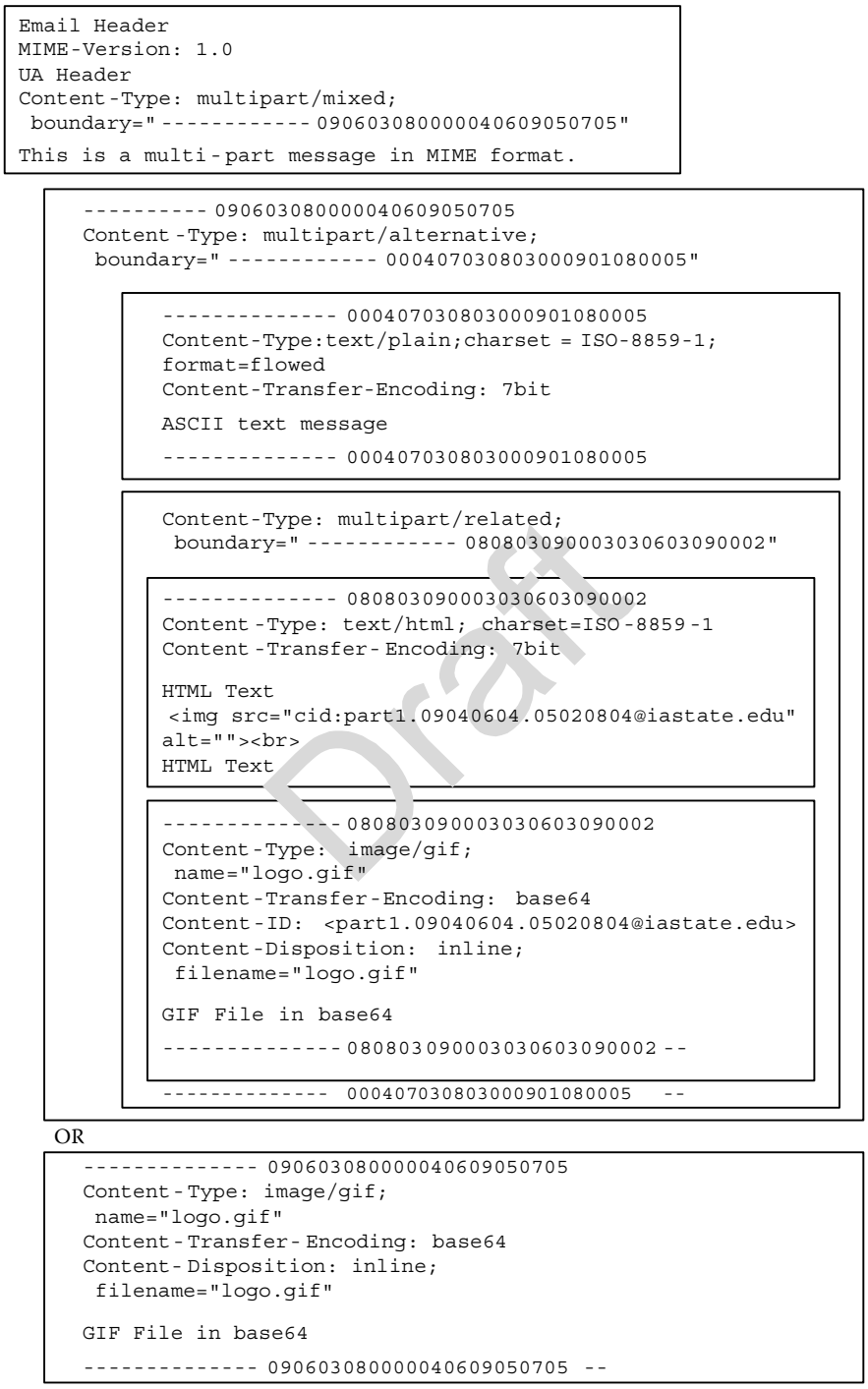


Figure 9.11: Multipart MIME message.

Content-Type: Multipart: Used to create documents that contain multiple objects. This is used to support attachments in email messages or to include multiple objects within a single email message. Figure 9.11 shows an email message with multiple parts. As we see in Figure 9.11, there are multiple MIME headers for each object type. This content type is used to transport malicious code, and depending on how the receiving user agent is set up, the malicious code may be executed when the email message is opened.

Content-Type: Message: Used if the object is a whole mail message or part of a mail message. The most interesting security aspect of this header is the ability to point to another message stored on a remote site. Again, depending on how the user agent is configured, remote messages can be used to transport malicious code. It can also be used to tell if an email message was opened. This is often referred to as an email bug, and will be discussed in the vulnerability section of this chapter.

The remaining content type headers are self-explanatory and do not represent additional security threats beyond what has already been described. They are methods to attach different file types and have the receiving user agent be able to process them.

Content-Transfer-Encoding: The content transfer encoding header indicates what data format is used to encode the objects contained in the message. The syntax of the header is:

```
Content-Transfer-Encoding: type
```

There are several types defined and this header does not represent a vulnerability since the encoding methods are simple translations.

Content-Id: The content id header is used to identify the message in a multiple-part message, and does not create any security threats.

Content-Description: The content description header is used to provide a description of the content. The syntax of the header is:

```
Content-Description: <description>
```

Depending on how the user agent presents this to the user, this could be used to mask the real identity of a file. For example, an attacker could provide a description of an attachment that indicates it is a JPEG picture file, but the content type header indicates it is an executable. If the user agent displays the content description to the user, then the user may try to open the file thinking it is picture when in reality the file could contain malicious code.

Content-Disposition: The content disposition header is used to tell the receiving user agent how to display the content. The syntax of the header is:

```
Content-Disposition: type
```

The two types of interest are inline and attachment. The inline type tells the user agent to display the object automatically in the message on the user's display. The attachment type indicates the object should be shown as an attachment to the user. The inline type could be used to help an attacker force the display of pictures or other objects in the output window. This could have two damaging side effects. If the picture or object contained malicious code, it could be used to attack the user agent when the email was opened. The second side effect is in spam emails, where the spammer wishes to embed his spam message as a picture to evade detection. These messages would be displayed and seen by the user.

There have been numerous addendums to the MIME protocol to support many additional file types. It is beyond the scope of this book to examine all aspects of the MIME protocol. From a security viewpoint, the MIME protocol provides an attacker with a method to directly access a naive user in order to get him or her to help with the attack. This makes email an interesting attack method since it can target both the application and the user.

9.3.1 Vulnerabilities, Attacks, and Countermeasures

There are not a large number of vulnerabilities in the MIME protocol. As we have seen, the MIME protocol enables attacks against the user. Many of the countermeasures are common across the categories of attacks. Probably the most

effective countermeasure to combat attacks against the user is awareness and training.

9.3.1.1 Header-Based Attacks

Invalid header-based attacks in MIME are handled by the user agent, and often the result is that the email message is not displayed. The biggest header-based vulnerability is that the headers can be used to hide the actual content of the message. As discussed earlier, there is a content description that can be used by the user agent to display the content type. This allows an attacker to create an email that claims to have a picture as an attachment, but really has an executable. Providing proper user education and training can help with this attack, and there are general email countermeasures that can mitigate some of these attacks.

Another method of attack is to use HTML-based email to hide the actual values that are in the mail messages. A mail message can contain a hyperlink to a web page. The text that a user clicks on to access the hyperlink can say anything. So a user can be fooled into clicking on a link and going to someplace other than where he thought he would end up. User education and training is the best way to mitigate this attack.

9.3.1.2 Protocol-Based Attacks

Since the MIME protocol does not involve interaction between the two layers, the protocol-based attacks are different than the protocol attacks we have discussed before. A MIME-based protocol attack uses the MIME protocol to attach malicious files. The user would activate the malicious code either by viewing the email or opening an attachment. This malicious code can attack other programs on the user's computer that are used to view the file. Many user agents have the ability to display the attached file directly, which is convenient for the user but can be a problem for security. There have been worms and viruses that have taken advantage of some user agents' ability to directly view attached data types. One such worm was the Nimda worm, which was launched by simply reading the email message. The worm then copied itself to the hard drive of the user's computer and emailed itself to users in the recipient's address book. Other user agents did not automatically open the attachment, but if the user opened the attachment, the machine became infected.

Typical countermeasures for protocol-based MIME attacks consist of disabling the features that allow direct viewing of attached data. Most user agents have a mode where they will ask before displaying inline content like pictures. There are technologies that will attempt to filter out malicious attachments. Another

countermeasure that has become common is to use host-based scanners and firewalls that can restrict the spread of the malicious code. A host-based firewall can prevent unauthorized programs from accessing the network, which works well against attached malicious code. However, if the user agent is the program that is spreading the malicious code, then the firewall typically will not stop it, since the user agent is an authorized user of the network. Again, since the MIME protocol directly interfaces with the user, it is important to educate the user on how to handle email attachments.

9.3.1.3 Authentication-Based Attacks

MIME does not directly support user authentication, but in a way it can enable spoofed authentication. MIME gives an attacker the ability to create convincing emails that look like they come from established organizations. One additional authentication-based attack is the use of email bugs to track where and when an email is opened. This can be done by inserting a picture into the email document, typically as part of an HTML document. The picture is 1×1 pixels in size and actually is stored on a remote web server. When the user reads the email message, the picture is downloaded from the remote web server. The remote web server logs the access from the user, providing the date and time, the IP address, and other information about the client software. The most common countermeasure is for the user agent to prompt the user before showing images on an email message. However, if the user always clicks yes, then this countermeasure is not effective.

9.3.1.4 Traffic-Based Attacks

MIME does not have any traffic-based vulnerabilities, but it does tend to create larger email messages, and with attachments can create massive email messages. The common countermeasure is to set a limit on the size of an email message that can be received by the MTA. This does not stop someone from sending a large number of smaller messages.

The threat of sniffing has not changed because of the MIME protocol. The countermeasures to network sniffing of email can be pushed to the user as described in the next section, or can be pushed on to the network as described later.

Definitions

Email bug.
A method of embedding a link to a web site inside an email message to enable the sender to see if the email was opened.

MIME transfer encoding.
The encoding method used by MIME to convert the non-ASCII data to ASCII text.
Multipurpose Internet Mail Extensions (MIME).
A message format used to support nontext content in email messages.

9.4 General Email Countermeasures

What we are looking for in a countermeasure is a way to authenticate the end users and to ensure that email messages are sent in such a way that they are unchanged, unread, and authenticated. There are some end user applications that can provide this kind of secure email.

Several vulnerabilities are caused by unauthenticated delivery of email, including spam email, phishing email, viruses, and other malicious code. There are several network-based technologies that can mitigate many of these attacks.

This section will examine several of these countermeasures and will also discuss methods to trace suspicious email messages or tell if an email is suspect.

9.4.1 Encryption and Authentication

Encryption has been used to prevent accidental or malicious viewing of data and to prevent sniffing of network traffic. Encryption can also be used to provide authentication of one or both communicating parties. Appendix A discusses several aspects of encryption that apply to network security. In the case of email, there are several possible places that encryption can be deployed, as shown in Figure 9.12.

Each of the possible encryption points shown in Figure 9.12 provides a different level of security, and each has its own set of problems. The first encryption point is the traffic between the MTAs. As we discussed earlier in the chapter, there have been proposals to encrypt traffic between the MTAs that would provide authentication of each MTA. This has been proposed as a method to reduce spam since only authorized MTAs can send email. From an implementation standpoint, this has several problems, including finding a way for each MTA to share the encryption keys (see Appendix A for a description of public keys and key distribution). The encryption keys are used as authenticators, and therefore they must be protected on the MTA, and the methods to distribute keys between MTAs must also be secure.

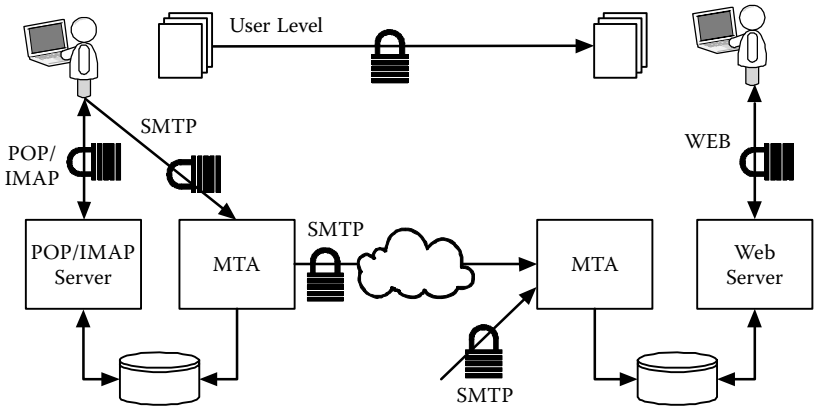


Figure 9.12: Possible encryption and authentication points.

There are several social and political issues that come into play when we require every MTA to be authenticated, including how to handle anonymous email. Another question is who decides which MTAs are trusted and how that process is managed. It is also unclear that such a system could actually keep out spam, since attackers could just take over trusted MTAs. Encrypting traffic between MTAs would stop the sniffing of any traffic between MTAs, which is possible at the egress points of an organization's network, but is very difficult across the Internet.

Another place where SMTP can be encrypted is between the user within a network and the MTA. This biggest benefit would be the authentication of the user in order to support the replaying of email. There is still the issue of key distribution, which can be handled using public keys. As discussed earlier, the most common way to handle this is using the IP address of the user agent.

The next point at which encryption could be deployed is between the MTA and the recipient's computer. As discussed earlier in the section, there are secure versions of POP and IMAP. These versions provide additional authentication using public keys and also protect the username and password from eavesdropping. These methods are quite effective, especially when remote access to the email is required. The methods used for this encryption are the same as those used in link-level encryption, as discussed in Chapters 5 and 6.

When the user accesses his email via a web site, that traffic can also be encrypted using the same methods that are employed by any secure web site. This method uses public keys and is built into the web browsers. This web-based encryption is discussed in the next chapter.

The encryption methods discussed so far have only protected the transmission of email between two devices and only provided authentication of the devices. What is really needed with email is the authentication of the sending and receiving users. In addition, if there is a concern about unauthorized viewing of email messages, then the email should be protected using end-to-end encryption.

One email issue that should be addressed is whether all email needs to be protected. An argument can be made that not all email needs to be protected, and that the identities of the sender and receiver are not critical or can be obtained via the message itself. However, there are cases where the email message might contain confidential information or we need to verify that only the intended recipient was able to read the message. Since the email system consists of multiple applications communicating using multiple protocols, the only logical method to provide end-to-end security and authentication is to rely on the user agents. The most common protocol is Pretty Good Privacy (PGP), which was developed by Philip Zimmerman in the early 1990s [24, 25]. PGP allows a user to create an email message that has been signed and encrypted so that the recipient is confident that the message came from a user that knew the private key of the sender, and the sender is confident that only a user that knows the private key of the receiving user can read the message. Figure 9.13 shows the PGP message structure and how an email message is signed and encrypted for transmission to the recipient. The following discussion assumes knowledge of basic encryption. The reader may wish to review Appendix A.

As shown in Figure 9.13, PGP feeds the user's message into a hash function. The hash value is encrypted using the private key of the sender, which is used to produce the digital signature. The digital signature is added to the message, and the signed message is then compressed and encrypted using symmetric key encryption. The encryption key is generated using a pseudorandom number generator. This one-time session key needs to be transmitted to the recipient in a manner that only the recipient can open. This is accomplished by encrypting the session key using public key encryption. The encryption key is the recipient's public key. The encrypted session key is attached to the encrypted message. The resulting message is converted to ASCII so it can be transmitted via email.

As shown in Figure 9.14, the process is reversed to extract the message. The inbound message is converted from ASCII to its binary form. The encrypted session key is extracted from the message. In addition to the encrypted session key, the message contains information to identify the intended recipient of the message. This allows a person to have multiple identities, each with different

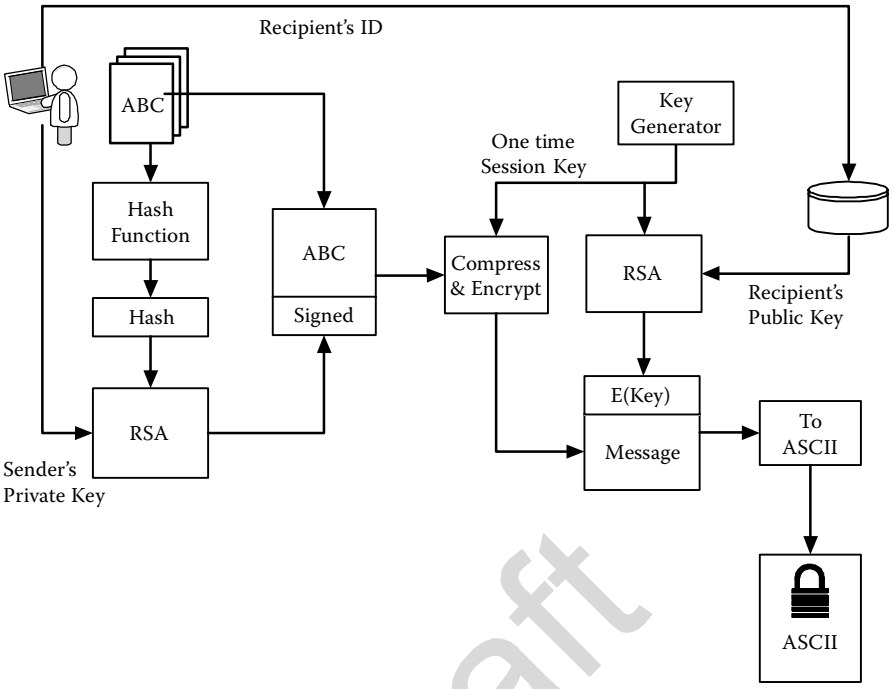


Figure 9.13: PGP message flow.

public–private key pairs. The id from the encrypted key field is used to index the private key. The private key of the recipient is used to decrypt the session key. The session key is used to decrypt the message. The message digital signature is extracted, and contained within the message digital signature field is an id that indicates the user who sent the message. That sender id is used to look up the sender's public key, which is used to decrypt the digital signature and extract the hash value. The message is then passed through the hash function and the two hash values are compared. If they are equal, then the message was successfully received.

What do we know about the sender and recipient after the message is successfully decrypted? We know from the digital signature that the message was created by someone that knows the private key of the sender. We also know that only a person that knows the private key of the recipient can successfully decrypt the message. The strength of this method is based on the level of protection used on the private keys.

PGP has had a moderate level of adoption. The strength of PGP has been well tested, and there have not been any major security issues. Its widespread adoption

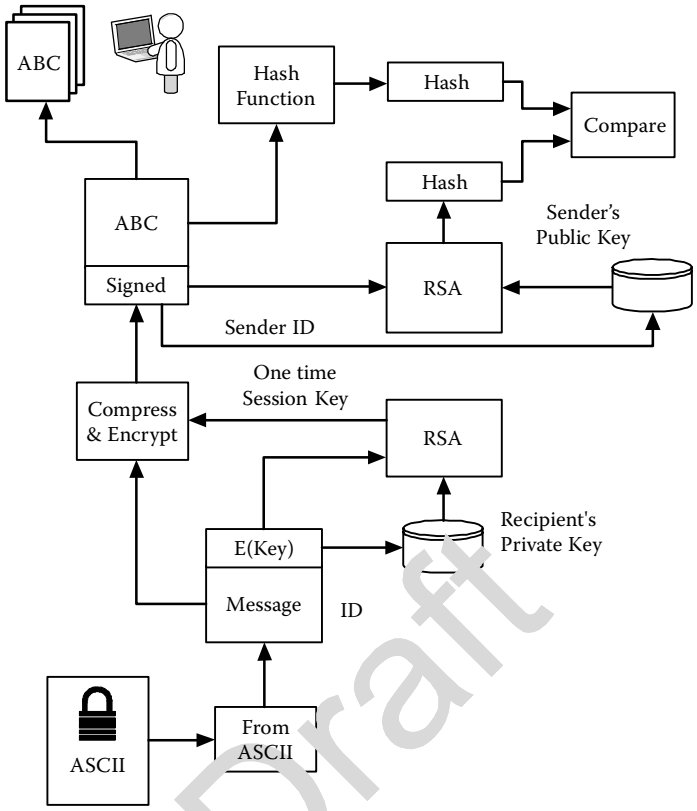


Figure 9.14: PGP message verification.

is hindered by issues with key distribution and key management. The primary issue with key distribution is how to know the owner of a public key and how to obtain someone's public key. There is no widely adopted method to distribute public keys and to make sure those keys represent the actual people. In addition, most people do not think their email is critical enough to warrant this level of security. However, PGP can solve the problems of eavesdropping and can be used to verify the sender and receiver of the email message.

9.4.2 Email Filtering

As we saw during the discussion of the MIME protocol, email can be used to transport malicious payloads and can be used for spam email and phishing attacks. There have been several proposed protocol modifications that have tried to address these issues; however, most have been unworkable. We have seen improvements

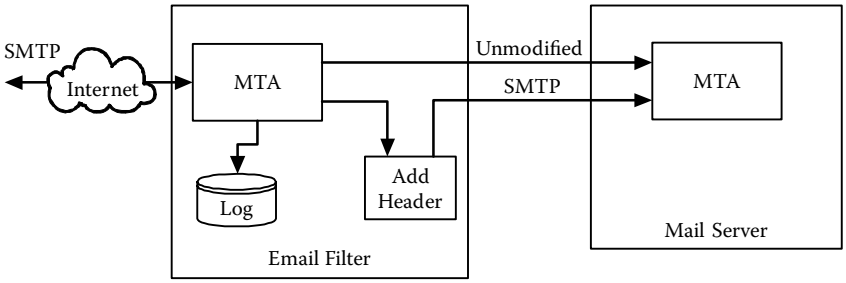


Figure 9.15: Email filtering.

in the user agents in an attempt to make them less susceptible to malicious code. One method that is gaining adoption is the use of email filters. An email filter typically sits in front of the email server and is the first MTA to receive the message. Depending on the filter type, the message can be passed on unmodified, modified to remove malicious payloads, or just deleted. Figure 9.15 shows a typical email filter and how it interacts with the organization's email server [26–37].

The inbound email arrives at the filter and is processed. The filter then forwards the email to the organization's email server, where it is handled like any other email message. Outbound email is forwarded from the organization's email server to the email filter, which will scan for outbound problems.

One type of email filter detects spam and phishing attacks. There are several approaches to handling spam email. The first involves trying to classify the email as spam by analyzing the message based on past message classification. The goal is to train the system to learn what spam email looks like. This type of classification has also been placed in most user agents. This allows the user to create a customized spam filter. These types of filters are not exact and can have false positives, which are when the email is classified as spam when it is not. A network-based spam classification filter will mark the message as spam by putting a line (spam marker) in the header of the email message; this way the user agent can trigger the spam marker and classify the message as spam. The user agent can be set up to mark the message as spam and display it with the nonspam messages, or it can automatically move the message into a spam folder. Since this method is not accurate, most organizations will not automatically delete messages based on the results of a classification filter.

Spammers keep adapting to new defense methods. One new method of bypassing the spam filters is to use the MIME protocol and not transfer any text, but create an email message that is a picture. The picture contains the advertisement

the spammers want the user to see. The classification systems cannot tell if the message is spam since they cannot analyze the contents of a picture. Another method often used is to spoof the sending address to make it something you would want to open. The spammers also create subject lines that make you want to open the emails, or use random subject lines to bypass the filters. All of these attacks are supported by the unauthenticated email system and the desire to have a user-friendly user agent.

Another spam filtering method (which also can work with other malicious emails) is using a filtering list. A filtering list can be used to reject email messages from a list of sites using the SMTP protocol. There is a question about what to use as the address for the filtering list. If you use the IP address of the sending MTA, then you will not stop email that has been forwarded through another MTA. You can use the sender's address information (username and domain name), but that can be spoofed. Even with these inherent problems, there are some benefits to email filtering lists. There are three types of email filtering lists.

The first type is called a blacklist and consists of banned email senders. The list can consist of domain names, user@domain, and IP addresses. The biggest problem with a blacklist is maintaining it. Email spammers are constantly changing their domains. There are web sites that maintain lists of banned sites, but this method only offers a small amount of relief.

A whitelist is the opposite of the blacklist and is very restrictive. The list contains the names or IP addresses of authorized email senders. This is useful for a private email system set up between several MTAs within an organization. Whitelists do not work for a public MTA, since the MTA has no way of knowing the users that will be sending mail beforehand.

The third type is called greylisting. Greylisting uses a feature of the SMTP protocol to stop robot-based spammers. A robot-based spammer uses a small application to create and send email messages to an MTA. The robot application does not implement all of the functionality of an MTA. A normal MTA will hold outbound email messages that have experienced a temporary failure when communicating with an MTA. The MTA will try to send the message again after waiting for some period of time and will continue to try for several days. The robot-based spammer will try to send a message, and if the receiving MTA sends a temporary failure message, it will quit and go on to the next destination.

A greylist filtering device will respond to the first email message from a new sender with a temporary failure message (SMTP response code of 451). If the sender sends the message again after a waiting time, then the filtering MTA will allow the message and add the sender to the greylist. The next time the sender

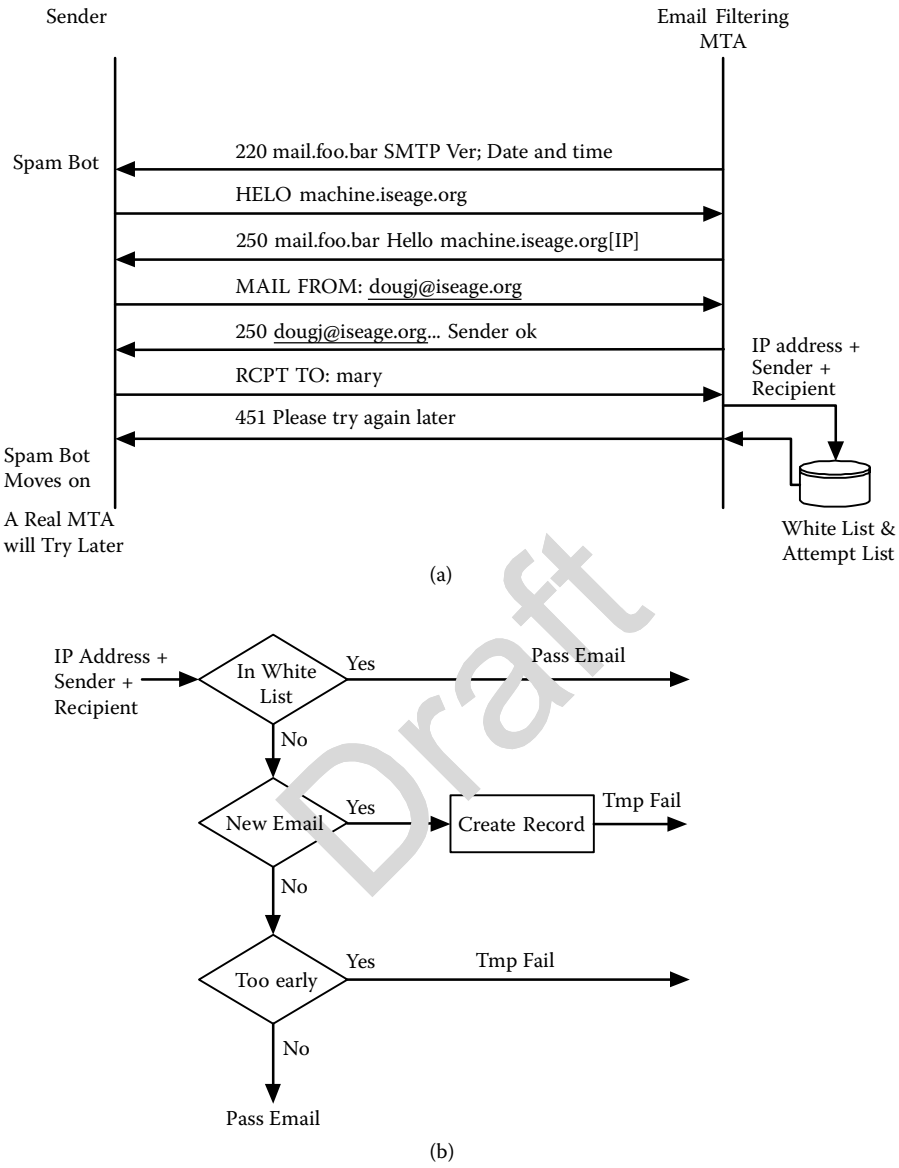


Figure 9.16: (a) Email greylisting. (b) Email greylist flowchart.

sends a message it will be allowed. Figure 9.16 shows how the greylist operates with both a real MTA and a spam robot.

As we see in Figure 9.16, the greylist also works with a whitelist, where the domains in the whitelist do not pass through the greylist process. The greylist

looks at the IP, sender's, and recipient's addresses to identify the email message. Greylisting can help reduce spam; however, the system can be defeated if the spammers resend the message after some waiting time or forward their spam through another MTA that is not using greylisting.

9.4.3 Content Filtering

Another method to help provide email security is to use an email content filtering device. This is different than the spam filter in that it will look for specific content that could cause a security problem. These devices do operate as an MTA and will analyze the content of the email messages to determine if they contain malicious payloads. The most common type is an email virus scanner, which scans all email messages for viruses. If a virus is found, the content filter would remove the offending message and allow the cleaned message to continue. It often adds a note in the message to indicate the email has been cleaned and can be configured to send a message back to the sender, informing her that her email contained a virus.

A content filter needs to understand the MIME protocol in order to decode the messages and scan them. In addition, most virus scanners will check both outbound and inbound email messages to prevent malicious payloads from leaving as well as entering the network. Since email is a store-and-forward application, the virus scanner has time to fully decode and check the email messages. These systems use a signature-based method to determine if the content contains a virus. The difficulty is in creating signatures that will detect only malicious code and not stop benign code. Another problem is keeping the signatures up to date with the viruses. There have been cases where several modified versions of the same virus have been released within a 24-hour period in an effort to evade the virus scanners.

Another type of content filter targets outbound content that should not leave a network. Government regulations have forced many organizations to install methods to prevent private data from leaving a network. Healthcare and financial records are protected by government regulations. An outbound email content filter will examine all outbound email messages looking for content that should not leave, such as social security numbers. This can be done through signatures or through exact content matching. Once an email has been tagged as containing private content, the filter can store the message and notify the sender about the violation. Some vendors have produced a system to encrypt the message before it leaves. They send the encrypted message to an off-site MTA that will send

notification to the recipient that private content is on this web site. The recipient would then log in to the web and retrieve the email using a secure web transaction. This method is good when we are only concerned about keeping others from reading the private data. If we are concerned about private data leaving the organization through email (either by accident or through a malicious act), then the best method is to quarantine the email.

Content filters have one weakness in that they cannot open or analyze encrypted email like PGP. For outbound content filters this is a problem for malicious data loss, but not a problem if your goal is to keep the email from being read by a third party. For virus scanners there have been cases where the attacker encrypts the virus and then attaches it to an email message. The body of the email explains to the recipient that for security reasons the attachment has been encrypted, and the attacker provides the encryption key in the message. This requires the user to take an additional step to infect the machine. Depending on the type of virus scanner, it may not be able to analyze compressed attachments since the signature would not be found in the data. As with most defense methods, attackers work hard to find ways to work around them.

9.4.4 Email Forensics

Since email has become one of the primary methods of attack against the users of a network, it is useful to understand how to read email messages in order to trace back a message to the sender. It is often not possible to trace the email back to the actual person, but you can trace the email back to the sending MTA [38]. Figure 9.17a to c shows the headers from various email messages. These email headers are from actual email messages; however, the names and IP addresses have been changed. As we discussed earlier, headers are added to the front of the mail message as each MTA processes the message; the exception to this is with spam filters. A spam filter will place information about the spam message in the MIME header section.

Figure 9.17a shows an email header that was sent from a web email system to a user. You can analyze the email headers from either the top or the bottom. Each device that touches the email adds a header. As shown in Figure 9.17a, four devices handled the email message and each placed a header in the message. Starting with the first device that placed header A on the message, we can see that it used the HTTP protocol to receive the message and the message is from Harry6502@spammer.fake and is destined to john@ee.mail.spam. Figure 9.17a also shows a picture that was derived from the headers depicting the path the

```
D  Received: from nf-out-0910.email.mta (nf-out- 0910.email.mta
   [192.168.182.188])
           by vulcan.ece.mail.spam (8.12.8/8.9.3) with ESMTP id
   k85FaxBT1486661
           for <john@ee.mail.spam>; Tue, 5 Sep 2006 10:36:59 -0500 (CDT)
C  Received: by nf-out-0910.email.mta with SMTP id p77so1381355nfc
            for <john@ee.mail.spam>; Tue, 05 Sep 2006 08:34:32 -0700 (PDT)
   DomainKey-Signature: a=rsa-sha1; q=dns; c=nofws;
           s=beta; d=spammer.fake;
           h=received:message-id:date:from:to:subject:mime-
   version:content-   type;
   b=BD9tHbNaozYZj9gNQqXmkrnHNA3N8+3W4NApcFJkKsKyX8DdOTS7Dp1VNunGx66SLcU5r
   YiDxCnY6SuVCktWq73DDH7MYEfWgaOtYdl/hILBIRVNcbLxGtyCoIT7I8use4F4RgCzZWc3
   Oc6fjqNzgGLe5s3RFQ9eVPhS+HxW+DA=
B  Received: by 10.65.237.1 with SMTP id o1mr4809264qbr;
           Tue, 05 Sep 2006 08:34:31 -0700 (PDT)
A  Received: by 10.70.63.2 with HTTP; Tue, 5 Sep 2006 08:34:31 -0700 (PDT)
   Message- ID:
   <ab156e9f0609050834v528b5b2eld9204458fe6409a1@mail.spammer.fake>
   Date: Tue, 5 Sep 2006 10:34:31 -0500
   From: "Harry Mudd" <Harry6502@spammer.fake>
   To: john@ee.mail.spam
   Subject: mail trace 2
   MIME-Version: 1.0
```

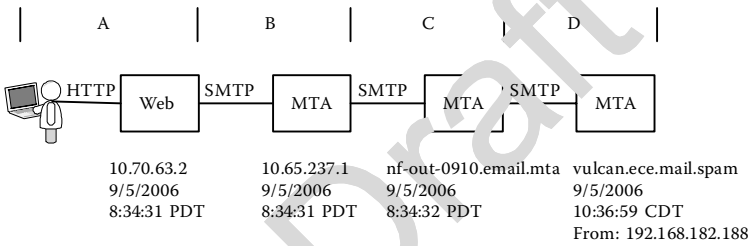


Figure 9.17a: Email headers.

emails took to reach the destination. The email headers also contain a time and date stamp when the message was received by each computer, as well as the machine name, IP address, or both. With this information we can tell where the message was sent from and what time it was sent. We cannot tell anything about the user agent other than it is a web-based user agent. By contacting the network registration authorities you can determine the owner of the IP address. The actual email message did use internal IP addresses for the first two machines, and therefore you cannot determine the location using the IP address. The MTA named "nf-out-0910.email.mta," however, did have a public IP address (in the original email message) and can be traced.

Figure 9.17b shows an email message that originates from a non-web-based user agent and passes through two different organizations with email filters. The transfer marked as A shows an email message being received by vulcan.ece.mail.spam

F
```
Received: from pop-5.mail.spam (pop-5.mail.spam [172.16.7.12])
        by vulcan.ece.mail.spam (8.12.8/8.9.3) with ESMTP id
k85FjSBT1508024
        for <john@EE.MAIL.SPAM>; Tue, 5 Sep 2006 10:45:28 -0500 (CDT)
```
E
```
Received: from devirus-2.mail.spam (devirus-2.mail.spam [172.16.7.10])
       by pop- 5.mail.spam (8.12.11.20060614/8.12.11) with SMTP id
k85Fgt28016542
        for <john@mail.spam>; Tue, 5 Sep 2006 10:42:55 -0500
```
D
```
Received: from (despam-3.mail.spam [172.16.7.5]) by devirus - 2.mail.spam
with smtp
         id 0df9_ae8af2c2_3cca_11db_969a_001372537fef;
       Tue, 05 Sep 2006 10:38:34 +0000
```
C
```
 Received: from magellan.sender.mta (magellan.sender.mta
[192.168.16.211])
        by despam- 3.mail.spam (8.12.11.20060614/8.12.4) with ESMTP id
k85FgttT020053
        for <john@mail.spam>; Tue, 5 Sep 2006 10:42:55 -0500
```
B
```
Received: from vulcan.ece.mail.spam (vulcan.ece.mail.spam [172.20.5.6])
        by magellan.sender.mta (8.13.6/8.13.6) with ESMTP id
k85Fgemo030599
        for <dwj@sender.mta>; Tue, 5 Sep 2006 10:42:40 -0500 (CDT)
       (envelope-from john@mail.spam)
```
A
```
Received: from [172.21.4.7] (babylon4.ece.mail.spam [172.21.4.7])
        by vulcan.ece.mail.spam (8.12.8/8.9.3) with ESMTP id
k85Fj6BT1501144
        for <dwj@sender.mta>; Tue, 5 Sep 2006 10:45:06 -0500 (CDT)
Message-ID: <44FD9AEC.4040103@mail.spam>
 Date: Tue, 05 Sep 2006 10:42:36 -0500
 From: Harry Mudd <Harry@mail.spam>
Organization: ISU Information Assurance Center
User-Agent: Mozilla Thunderbird 1.0.7 (Windows/20050923)
X-Accept-Language: en-us, en
MIME-Version: 1.0
 To: Dave Johnson <dwj@sender.mta>
Subject: test 4
Content-Type: text/plain; charset=ISO-8859-1; format=flowed
Content-Transfer-Encoding: 7bit
```
Spam Filters
```
X-Filter-MailScanner- Information: Please contact the ISP for more
information
X-Filter-MailScanner: Found to be clean
X-Filter-MailScanner-SpamCheck: not spam, SpamAssassin (score=-2.6,
        requ ired 6, autolearn=not spam, BAYES_00 - 2.60, SPF_PASS -0.00)
X-Filter-MailScanner-From: john@mail.spam
X-PMX-Version: 5.2.0.264296, Antispam-Engine: 2.4.0.264935, Antispam-
Data: 2006.9.5.82442
X-Perlmx-Spam: Gauge=IIIIIII, Probability=7%, Report='__C230066_P5 0,
 __CP_URI_IN_BODY 0, __CT 0, __CTE 0, __CT_TEXT_PLAIN 0, __HAS_MSGID 0,
__MIME_TEXT_ONLY 0, __MIME_VERSION 0, __SANE_MSGID 0, __USER_AGENT 0'
```

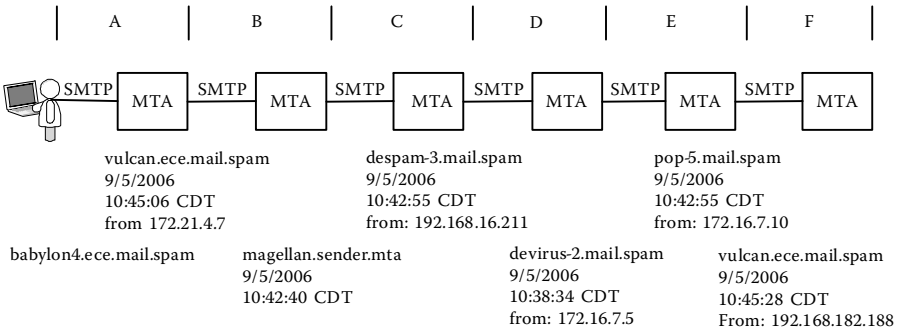


Figure 9.17b: Email headers.

```
(Removed local headers)
D  Received: from ns09.egujarat.net (202-149-46-162.static.exatt.net
   [202.149.46.162] (may be forged))
          by despam-2.iastate.edu (8.12.11.20060614/8.12.4) with ESMTP id
   k89KIRCr017274
          for <dougj@iastate.edu>; Sat, 9 Sep 2006 15:18:28 -0500
C  Received: from ns09.egujarat.net (localhost.localdomain [127.0.0.1])
          by ns09.egujarat.net (8.13.5/8.13.5) with ESMTP id
   k89H5sYI007263
          for <dougj@iastate.edu>; Sat, 9 Sep 2006 22:37:19 +0530
B  Received: (from administrator@localhost)
          by ns09.egujarat.net (8.13.5/8.13.5/Submit) id k89Gxf4q006335;
          Sat, 9 Sep 2006 22:29:41 +0530
A  Date: Sat, 9 Sep 2006 22:29:41 +0530
   Message-Id: <200609091659.k89Gxf4q006335@ns09.egujarat.net>
   To: dougj@iastate.edu
   Subject: Password change required!
   From: "eBay Inc." <admin@eBay.com>
   Content-Type: text/html
Spam Filter 2
   X-egujarat-MailScanner-Information: Please contact the ISP for more
   information
   X-egujarat-MailScanner: Found to be clean
   X-MailScanner-From: administrator@ns09.egujarat.net
Spam Filter 1
   X-PMX-Version: 5.2.0.264296, Antispam-Engine: 2.4.0.264935, Antispam-
   Data: 2006.9.9.124943
   X-Perlmx-Spam: Gauge=XXXXXXXXXIIIIIIIII, Probability=99%,

   <p><img src="http://pics.ebaystatic.com/aw/pics/navbar/eBayLogoTM.gif"
   width="150" height="70"></p>
   <BR>
Logo                Dear sir, <BR>
                    <BR>
                     We recently have determined that different computers
   have logged onto your eBay account, and multiple
    password failures were present before the logons. We strongly advice
   CHANGE YOUR PASSWORD. <BR>
                    <BR>
                     If this is not completed by <STRONG>September 15,
   2006</STRONG>, we will be forced to suspend your
    account indefinitely, as it may have been used for fraudulent purposes.
   Thank you for your cooperation. <BR>
                    <BR>
Phishing Site  <A
   href="http://linux.net zero.idv.tw/~ming/.change/index.php?MfcISAPIComma
   nd=ChangeFPP"
   target=_blank>Click here to Change Your Password</A></TD>
```

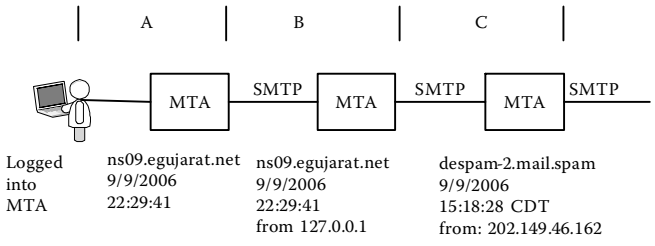


Figure 9.17c: Phishing email.

from a machine called babylon4.ece.mail.spam that is destined for the user dwj@sender.mta. Notice that the user agent that created the email message left information in the header. We know the user agent was Thunderbird running on a Windows machine, and we know the name of an organization that was configured in the user agent.

The transfer marked as B shows the message arriving at the destination. The message is then sent to despam-3.mail.spam and is destined for a different user. The email was forwarded so the user dwj sent his email to the user john on the machine mail.spam. The mail then follows a path through a spam filter and a virus filter, finally arriving at the machine vulcan.ece.mail.spam. Even though this email came back to the sending machine, you can see how email messages can be traced.

The spam filters added headers in the MIME header section. A user agent can be configured to take action based on the values found in these fields. Notice that the email was sent through two different spam and virus scanners. The first four lines (each starting with X-Filter) were added by one of the spam filters, and it determined the email was not spam and had no viruses. The lines starting with X-PMX and X-Perlmx are from a different spam filter, and it also determined the message did not contain spam. In some cases the antispam filter will also add text in front of the subject line to help the user tell if a message is spam.

The last email message, shown in Figure 9.17c, is an actual spam message with the sending addresses in tact. The recipient's addresses have been changed. This message also shows using MIME to create an email message to trick the user into going to a web site and entering her account information. This is known as phishing.

As we see in Figure 9.17c, in header A the sender's domain (ebay.com) does not match the name of the first MTA (ns09.egujarat.net). Headers B and C indicate that the sender of the email was logged into the machine that created the message. The IP address of ns09.egujarat.net was 221.128.130.1, which is not the IP address of the machine connecting to despam-2, as shown in header D. This indicates the machine name was spoofed by the sender. The message came from an IP that belongs to an ISP. Also notice that two spam filter headers have been added. Spam filter header 1 was added by the recipient's network, and spam filter header 2 was added by the spammer.

Figure 9.17c also shows how an image is copied from another web site to show up in the email. In this case, a logo from eBay is included in the email. The email also contains a link to a web site that will carry out the phishing attack. Parts of the message body have been removed to save space.

There are commercial tools that help trace back an email message. These tools just automate what can be done by hand.

Definitions

Email blacklist.
A method of filtering email where the filter maintains a list of bad email servers, called a blacklist.

Email content filter.
A method of filtering email where the filter examines the contents of the email and determines if the email should be filtered, altered, or quarantined.

Email filter.
A device that examines email messages and determines if the email should be delivered or modified based on a set of rules.

Email forensics.
A process of analyzing email headers to determine the actual sender of the email messages.

Email greylist.
A method of filtering email where all unknown incoming email messages are temporally rejected until they retry. This is designed to stop programs that send email messages without the use of an MTA.

Email phishing.
Email messages that are designed to trick the reader into providing information to the attacker.

Email whitelist.
A method of filtering email where the filter maintains a list of good email servers that it will only accept email from, called a whitelist.

PGP email.
A method to encrypt and sign email messages between users.

Homework Problems and Lab Experiments

Homework Problems

1. Assume a user is sending a message containing only the word “hello” to the email server dougj.net.

a. Show the protocol exchange required to send the word "hello" to dougj@dougj.net from john@issl.org. Include the TCP open and close exchange.

b. Estimate the total number of bytes (TCP payload) required to send this message. Assume each message is sent as one TCP packet.

c. What is the total number of bytes transmitted on the network, including all packets?

d. What is the total overhead (total number of bytes sent on the wire versus the total size of the payload)?

e. What is the total overhead (total number of bytes sent on the wire versus the total size of the user message)?

2. Research the 1988 Internet worm and how it propagated.

3. MTAs are unauthenticated. What effect would requiring user authentication to send email have on the email system?

4. Why are POP and IMAP protocols authenticated services and the SMTP service unauthenticated?

5. Research the various methods that have been proposed to reduce spam email and comment on why they have not been effective.

6. Research methods spammers use to bypass spam filters and comment on how effective they have been.

7. Research email virus scanners and compare them to how host-based virus scanners work.

8. Research the methods attackers are using to bypass virus scanners to get users to run malicious code. Theorize some mitigation methods.

9. There are several different encryption methods that have been proposed for email, as was shown in Figure 9.12. They can be divided into three types: SMTP encryption, POP/IMAP encryption, and user-to-user encryption. Comment on the pros and cons of each one. If you had to pick just one to provide email security, which would it be and why?

10. Email was originally designed to be used to send simple text messages. MIME has enabled the transfer of complex data types. Comment on the security versus usability of having MIME-enabled email.

11. Research and find web sites that monitor the spam bots.

12. What type of information in headers can be used to trace an email message back to the sender? Comment on how useful the traceback would be in finding someone that is using email for illegal purposes.

13. Examine the headers of spam email you have received in the past few days. Build a table showing how many messages came from each country (if you can tell).

Lab Experiments

1. Use TELNET to connect to the email server in the test lab. Send an email message to your user account pretending to be anyone. You will need to enter the SMTP commands by hand.

2. Use TELNET to connect to the POP3 server on the same computer you used in experiment 1. Retrieve the email message you sent yourself in experiment 1 using the POP3 commands.

3. Find an email traceback website on the Internet. Take several email messages you have received and look at the results of the traceback.

4. Use tcpdump or wireshark to capture the sending of an email message between two machines in the test lab. Try to capture SMTP, POP, and IMAP.

Programming Problems

1. Download the file spam.tar from ftp://www.dougj.net. This file contains the core code to interact with an MTA using SMTP. There is a description of the program in spam.tar located on the web site. Perform the following:

    a. Modify the program to send email to your account on the email server in the test lab from a nonexistent user. Use the parameter (-s user@host) to pass in the fake source address and the parameter (-u user) to pass in the username of the target.

    b. Log in to the email server and view your email.

    c. Modify the program to accept a file as a parameter (-f filename) and send the file as email.

    d. Create an HTML file with the MIME headers and send it yourself using the modified spam program. Log in to the server to read the email.

2. Use the code you downloaded for Chapter 5 and modified in Chapter 6. Add code to perform the following:

   a. Decode and print SMTP payload. Print the payload in ASCII.

   b. Decode and print POP payload. Print the payload in ASCII.

   c. Decode and print IMAP payload. Print the payload in ASCII.

   d. Add to the set of counters a counter for the number of SMTP, POP, and IMAP packets. Add code to print the values of these counters to the subroutine program_ending().

References

[1] Leiner, B. M., et al. 1999. A brief history of the Internet. Arxiv preprint cs.NI/9901011.

[2] Segal, B. 1995. A short history of Internet protocols at CERN. http://www.cern.ch/ben/TCPHIST.html, accessed August 23, 2008.

[3] Mowery, D. C., and T. Simcoe. 2002. Is the Internet a US invention? An economic and technological history of computer networking. *Research Policy* 31:1369–87.

[4] Leiner, B. M., et al. 1997. The past and future history. *Communications of the ACM* 40:103.

[5] Hafiz, M. 2005. Security patterns and evolution of MTA architecture. In *Conference on Object Oriented Programming Systems Languages and Applications*, San Diego, CA: 142–43.

[6] Giencke, P. 1995. The future of email or when will grandma be on the net? In *Electro/95 International. Professional Program Proceedings*, Boston, MA: 61–67.

[7] Knowles, B., and N. Christenson. 2000. Design and implementation of highly scalable e-mail systems. Paper presented at Proceedings of the LISA Conference, New Orleans, December.

[8] Freed, N., and N. Borenstein. 1996. *Multipurpose Internet mail extensions (MIME) part one: Format of Internet message bodies*. RFC 2045.

[9] Freed, N., and N. Borenstein. 1996. *Multipurpose Internet mail extensions (MIME) part two: Media types*. RFC 2046.

[10] Moore, K. 1996. *MIME (multipurpose Internet mail extensions) part three: Message header extensions for non-ASCII text*. RFC 2047.

[11] Freed, N., J. Klensin, and J. Postel. 1996. *Multipurpose Internet mail extensions (mime) part four: Registration procedures*. RFC 2048.

[12] Freed, N., and N. Borenstein. 1996. *Multipurpose Internet mail extensions (MIME) part five: Conformance criteria and examples*. RFC 2049.

[13] Postel, J. B. 1982. *SMTP-simple mail transfer protocol*. RFC 821. http://www.ietf.org/rfc/rfc0821.txt, accessed August 23, 2008.

[14] Leiba, B., et al. 2005. SMTP path analysis. Paper presented at Proceedings of the Second Conference on E-mail and Anti-Spam (CEAS). Stanford, CA.

[15] Secure, S. 2002. Network working group p. Hoffman request for comments: 3207 Internet mail consortium obsoletes: 2487 February category: Standards track.

[16] Hoffman, P. 1999. *SMTP service extension for secure SMTP over TLS*. RFC 2487.

[17] Hoffman, P. 2002. *SMTP service extension for secure SMTP over transport layer security*. RFC 3207.

[18] Manabe, D., S. Kimura, and Y. Ebihara. 2006. *A compression method designed for SMTP over TLS*, 803. Lecture Notes in Computer Science 3961.

[19] Gray, T. 1993. Comparing two approaches to remote mailbox access: IMAP vs. POP, 1–4. http://www.imap.org/imap.vs.pop.brief.html, accessed August 23, 2008.

[20] Newman, C. 1999. *Using TLS with IMAP, POP3 and ACAP*. RFC 2595.

[21] Crispin, M. 1996. *Internet message access protocol—version 4rev1*. RFC 2060. Sebastopol, CA.

[22] Garfinkel, S. 1995. *PGP: Pretty good privacy*. O'Reilly.

[23] Garfinkel, S. L., et al. 2005. How to make secure email easier to use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Portland, OR: 701–10.

[24] Zhou, D., et al. 1999. Formal development of secure email. Paper presented at Proceedings of the 32nd Annual Hawaii International Conference on System Sciences. Maui, HI.

[25] Borisov, N., I. Goldberg, and E. Brewer. 2004. Off-the-record communication, or, why not to use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, Washington, DC: 77–84.

[26] Hidalgo, J. M. G. 2002. Evaluating cost-sensitive unsolicited bulk email categorization. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, Madrid, Spain, 615–20.

[27] Michelakis, E., et al. 2004. Filtron: A learning-based anti-spam filter. Paper presented at Proceedings of the First Conference on Email and Anti-Spam (CEAS). Mountain View, CA.

[28] Bass, T., and G. Watt. 1997. A simple framework for filtering queued SMTP mail (cyberwar countermeasures). In *MILCOM 97 Proceedings*, Monterey, CA: 3.

[29] Cerf, V. G. 2005. Spam, spim, and spit. *Communications of the ACM* 48:39–43.

[30] Jung, J., and E. Sit. 2004. An empirical study of spam traffic and the use of DNS black lists. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, Taormina, Sicily, Italy. 370–75.

[31] Golbeck, J., and J. Hendler. 2004. Reputation network analysis for email filtering. Paper presented at Conference on Email and Anti-Spam (CEAS). Mountain View, CA.

[32] Kartaltepe, E. J., and S. Xu. 2006. Towards blocking outgoing malicious impostor emails. In *Proceedings of the 2006 International Symposium on World of Wireless, Mobile and Multimedia Networks*, Buffalo, NY: 657–61.

[33] Gansterer, W. N., A. G. K. Janecek, and P. Lechner. 2007. A reliable component-based architecture for e-mail filtering. In *Proceedings of the Second International Conference on Availability, Reliability and Security*, 43–52.

[34] Twining, R. D., et al. 2004. Email prioritization: Reducing delays on legitimate mail caused by junk mail. In *Proceedings of Usenix Annual Technical Conference*, Boston, MA: 45–58.

[35] Levine, J. R. 2005. Experiences with greylisting. Paper presented at Conference on Email and Anti-Spam. Stanford University, Stanford, CA.

[36] Wiehes, A. 2005. Comparing anti spam methods. Masters of Science in Information Security, Department of Computer Science and Media Technology, Gjøvik University College.

[37] Miszalska, I., W. Zabierowski, and A. Napieralski. 2007. Selected methods of spam filtering in email. In *CADSM'07. 9th International Conference: The Experience of Designing and Applications*, Chapel Hill, NC: 507–13.

[38] de Vel, O., et al. 2001. Mining e-mail content for author identification forensics. *ACM SIGMOD Record* 30:55–64. Santa Barbara, CA.

Chapter 10

Web Security

The World Wide Web (WWW) is more than a collection of protocols [1–7]. The web consists of a large number of servers that are each identified by a host name. Each server contains documents that can be accessed using the document's address. The web has had the largest impact on the Internet and has driven many of the newest technological changes. Primarily because of the web we now have Internet access almost everywhere. The web has taken the Internet from a network used by researchers and academicians to a network that is used by the masses.

Because of the large number of servers and the large number of users, the web has also become a primary target of hackers. Before we begin looking at the network protocols, we need to understand the basic structure of the web and the applications that support it. Figure 10.1 shows how a document is addressed within the World Wide Web.

As we see in Figure 10.1, the user provides an address to a document using the host name of the server and the location of the document within the server. This address is called the Uniform Reference Locator (URL). The URL uniquely identifies a document within the web. Documents can contain links to other documents called hyperlinks. These hyperlinks are also URLs. A web designer uses hyperlinks to create a path or series of paths that provide a way for the user to navigate through the documents stored on the web server. The web was not designed to have a central index to keep track of the location of documents, which is the reason for the popularity of search engines. A search engine visits web sites and examines the documents and catalogs their contents. The search engines follow the hyperlinks to gather additional content. The information that is gathered is searched to provide answers to the user's query. The search engines are web sites that produce a web document with hyperlinks to the documents that match the user's query.

Figure 10.2 shows how one hyperlink can provide the location of a series of documents. The user can access a search engine to get the location of the first web site, or the user may know what site he or she wishes to visit. Figure 10.2 shows the user accessing the document D1, which contains links to other sites, on the server S1 using the URL HTTP://S1/D1. The user goes to another site by

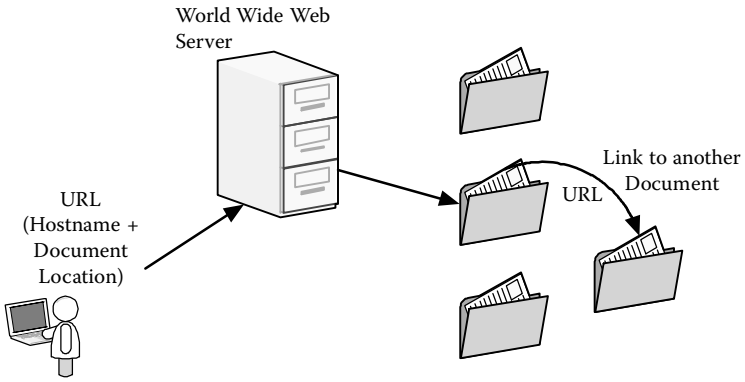


Figure 10.1: Document addressing in the World Wide Web.

just clicking on the hyperlink. In addition, a site may contain content that comes from yet another site, which is accessed when the user views the site. So, for example, document D2 on server S2 may contain a picture stored on server S3. This highly distributed nature of the web provides access to vast amounts of data. The distributed nature also provides many ways for attackers to compromise the data or the servers.

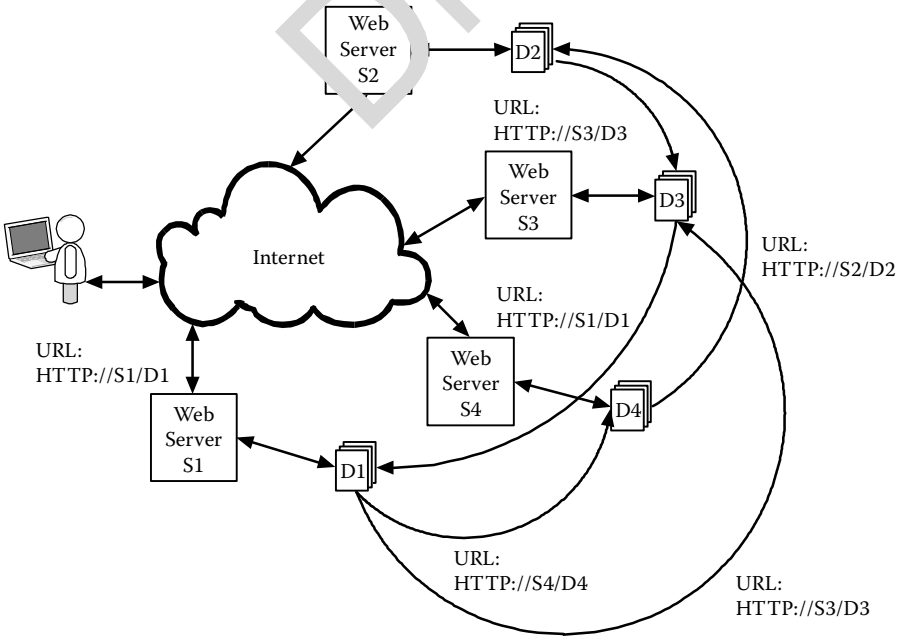


Figure 10.2: Hyperlinked documents in the World Wide Web.

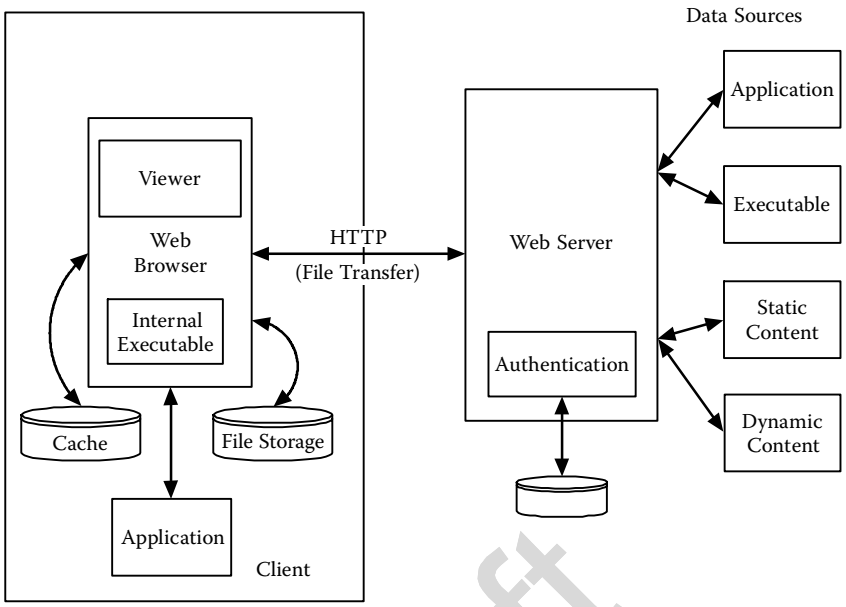


Figure 10.3: Web client/server.

The web was first designed for text-based data and provided access to library materials. As computers became more powerful and graphical user interfaces became more common, the content available on the web changed. The modern web involves a graphical client and a web server, as shown in Figure 10.3.

As we see in Figure 10.3, the client used to access the web is called the browser, and it consists of several parts. It has a viewer that interprets the data received from the server and displays it for the user. The user can also download documents and store them on the user's computer. In addition, it has a cache to store images and other documents received from the servers. The browser also has the ability to run applications to help the user interpret the data; for example, it can launch a media player to play a song. The browser can also run programs provided by the server, which allows for animations and complex interaction with the server. The browser uses a file transfer protocol called Hypertext Transfer Protocol (HTTP) to move data to and from the web server.

The web server sends files identified by a unique URL, requested by the browser, using the HTTP protocol. The requested documents can come from several sources, as shown in Figure 10.3. The most common are documents that are simple text files, whose content is static. Dynamic documents are produced by

a program running on the server that interprets the URL and creates a document from gathered or stored data that is sent back to the browser. For example, the document produced by a search engine as the result of a user's query is dynamically created from the search results. In addition to static and dynamic documents, the browser may request data from an application that is running on the server or from a program initiated by the server. The server also can provide user authentication to restrict access to certain documents.

We will first examine the HTTP protocol and look at its security vulnerabilities, then look at the document format. Since both the client and server can execute code, we will examine the server-side and client-side executables, and finally we will discuss several general network-based countermeasures used for web security.

10.1 Hypertext Transfer Protocol (HTTP)

The HTTP protocol is a command-and-response protocol with ASCII-based commands [8–11]. The structure is more complex than what we have seen with the email protocols. The basic message structure for the commands and responses is shown in Figure 10.4.

10.1.1 Command Message

As shown in Figure 10.4, the command message starts with a request line that has a request type, a URL, and the HTTP version. The request is:

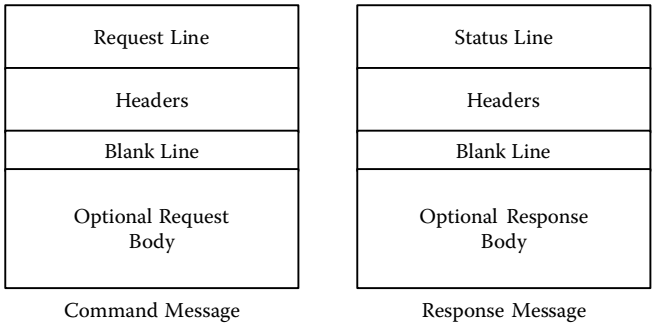


Figure 10.4: HTTP command and response message structure.

Format	request type<sp>URL<sp>HTTP/version
Example	GET http://www.iseage.org HTTP/1.1

GET is a request type used to request a document specified by the URL (http://www.iseage.org) and the protocol version is HTTP/1.1. The format of a URL is:

```
method://host[:port][/path]
```

The method indicates the file transfer protocol that is used to obtain the document. The default is HTTP. Browsers may support other file transfer types such as File Transfer Protocol (FTP). The optional port number is the server port used for the connection. The default for HTTP is port 80. The optional path is the location of the document on the server. If no path is given, the server will return a site-specific default document. The HTTP protocol supports several request types, as shown in Table 10.1. As we have seen with other protocols, many of the request types are often disabled for security reasons (as indicated in Table 10.1).

TABLE 10.1: HTTP Request Types

Type	Action
GET	Retrieve a document specified by the URL.
HEAD	Retrieve the headers from the document specified by the URL (response does not contain the body).
POST	Provide data to the server.
PUT	Provide new or replacement document specified by the URL (disabled).
PATCH	Provide differences to document specified by the URL in order to change the document (disabled).
COPY	Copy the document specified by the URL to the file specified in the header (disabled).
MOVE	Move the document specified by the URL to the file specified in the header (disabled).
DELETE	Delete the document specified by the URL (disabled).
LINK	Create a link to the document specified in the URL. The name of the link is specified in the header (disabled).
UNLINK	Remove the link specified in the URL (disabled).
OPTION	Ask the server what options are available.

TABLE 10.2: Response Code Format

Code	Response Status
1XX	Information messages
2XX	Successful requests
3XX	Redirect the client to another document specified as a URL
4XX	Client-side errors
5XX	Server-side errors

10.1.2 Response Message

Just as the commands are in ASCII, response messages are also in ASCII. The response message starts with a status line that consists of the HTTP version, a three-digit ASCII number, or response code, and a status phrase as shown below:

Format	HTTP/version<sp>status code<sp>status phrase
Example	HTTP/1.1 404 File not found

The first digit of the response code (404 in the above example) indicates whether the command worked or failed, the second digit specifies what type of code (typically 0), and the third digit is used to indicate specific codes. The response code syntax is shown in Table 10.2, and Table 10.3 shows several common response codes.

10.1.3 HTTP Headers

The next part of the HTTP request and response messages is the headers. As shown in Figure 10.5, the header format for request and response messages is the same. It should be noted that the headers can be optional, especially in the request message.

The HTTP header consists of one or more lines of text, each starting with a header name, followed by a colon, a space, and the header values as shown below:

Header Name:<sp>Header Value

The general header contains information about the request or response. The common general headers are shown in Table 10.4.

The request header is used in the request message. This header provides the server information about the client's configuration and indicates the preferences

TABLE 10.3: Common Response Codes

Code	Phrase	Meaning
100	Continue	First part of the request has been received. The client can continue.
200	OK	Successful request
204	No content	The body contains no content.
302	Moved permanently	The document specified by the URL is no longer on the server.
304	Moved temporarily	The document specified by the URL has temporarily moved.
400	Bad request	The request contained a syntax error.
401	Unauthorized	The authentication failed for the requested document.
403	Forbidden	The service requested is not allowed.
404	Not found	The document requested is not found.
405	Method not allowed	The method requested in the URL is not allowed.
500	Internal server error	The server failed.
501	Not implemented	The requested action cannot be preformed by the server.
503	Service unavailable	The request cannot be accomplished right now; try again later.

the client has about the format of the documents. From a security standpoint, this header can provide the server with information about the browser and the user. The common request headers are shown in Table 10.5.

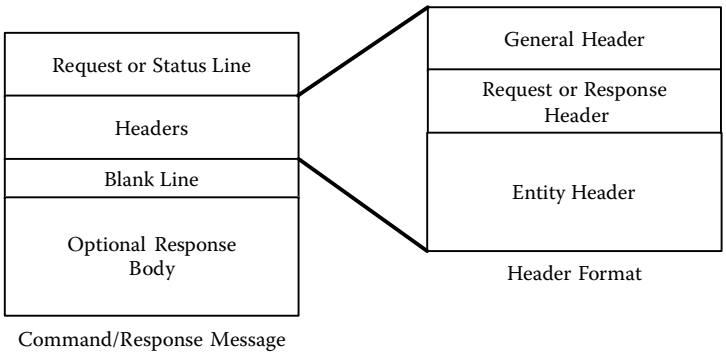


Figure 10.5: HTTP message header format.

TABLE 10.4: Common General Headers

Header	Function
Cache-control	Used to specify information about the client-side cache
Connection	Indicates whether the connection should be closed
Date	Provides the current date
MIME-version	Indicated the MIME version being used
Connection	Use to determine connection type
Keep-alive	Used to manage keep-alive connection

The response header reports back information about the server and about the requested document, as shown in Table 10.6.

The entity header contains information about the data contained in the body of the message, such as the type of encoding and the length of the data. The common entity headers are shown in Table 10.7.

An example exchange between a web browser and a web server is shown in the next several figures. Figure 10.6 shows the web page that was retrieved by the browser, and Figure 10.7 shows a summary of the packets used to load the web page.

Figure 10.8 shows the request message, and Figure 10.9 shows the response message. Figure 10.10 shows the request for the first image, and Figure 10.11 shows the response. Figures 10.12 and 10.13 show the packet exchange to retrieve the file favicon.ico, which is the small picture shown next to the URL in the browser window. This web site did not have the favicon.ico file, so the web server returned an error.

TABLE 10.5: Common Request Headers

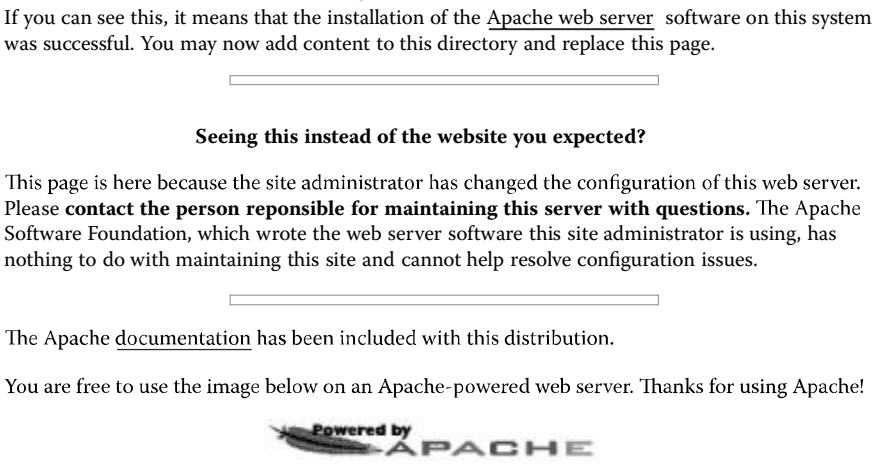
Header	Function
Accept	Indicates which data formats the browser can accept
Accept-charset	Indicates the character set(s) the browser can accept
Accept-encoding	Indicates what encoding methods the browser can process
Accept-language	Indicates what language the browser can accept
From	Provides the e-mail of the user on the browser
Host	Provides the host and ephemeral port of the browser
Referrer	Provides the URL of the linked document
User-agent	Provides information about the browser software

TABLE 10.6: Response Header

Header	Function
Accept-range	Indicates the server accepts the range requested by the browser
Retry-after	Indicates the date when the server will be available
Server	Provides the server application name and version

TABLE 10.7: Common Entity Headers

Header	Function
Allow	Provides a list of methods allowed for the URL
Content-encoding	Indicates the encoding method for the document
Content-language	Indicates the language of the document
Content-length	Indicates the length of the document
Content-location	Real name of the document requested
Content-type	Indicates the media type of the document
Etag	Provides a tag for the document
Last-modified	The date the document was last modified


If you can see this, it means that the installation of the Apache web server software on this system was successful. You may now add content to this directory and replace this page.

Seeing this instead of the website you expected?

This page is here because the site administrator has changed the configuration of this web server. Please **contact the person reponsible for maintaining this server with questions.** The Apache Software Foundation, which wrote the web server software this site administrator is using, has nothing to do with maintaining this site and cannot help resolve configuration issues.

The Apache documentation has been included with this distribution.

You are free to use the image below on an Apache-powered web server. Thanks for using Apache!

Figure 10.6: Web page image.

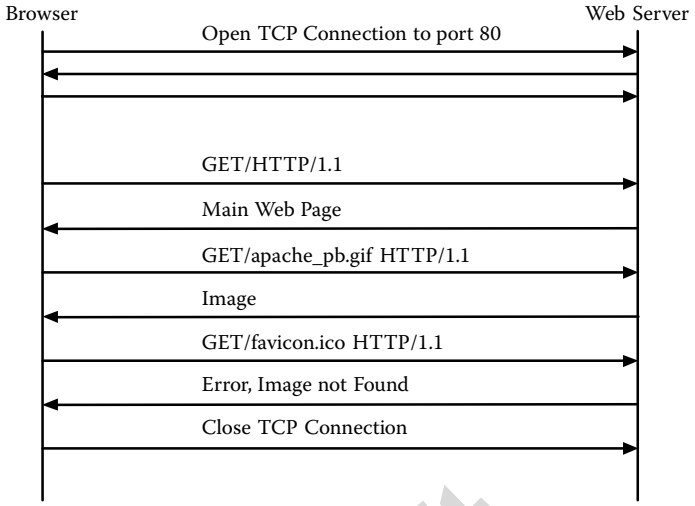


Figure 10.7: HTTP protocol exchange.

Request Line

GET/HTTP/1.1

General Header

Keep-Alive: 300
Connection: keep-alive

Request Header

Host: spock.ee.iastate.edu
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.7) Gecko/20060909 Firefox/1.5.0.7
Accept: text/xml, application/xml, application/xhtml + xml, text/html; q = 0.9, text/plain; q = 0.8, image/png, */*; q = 0.5
Accept-Language: en-us, en; q = 0.5
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1, utf-8; q = 0.7, * ; q = 0.7

No Entity Header
Blank Line
No Body

Figure 10.8: HTTP request message.

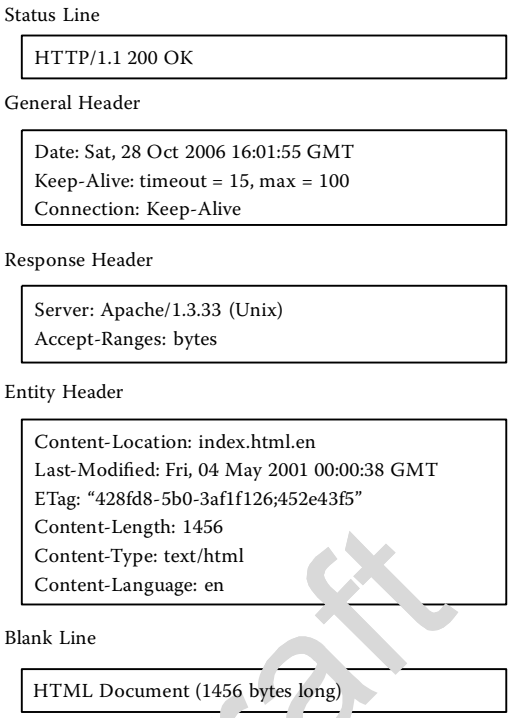


Figure 10.9: HTTP response message.

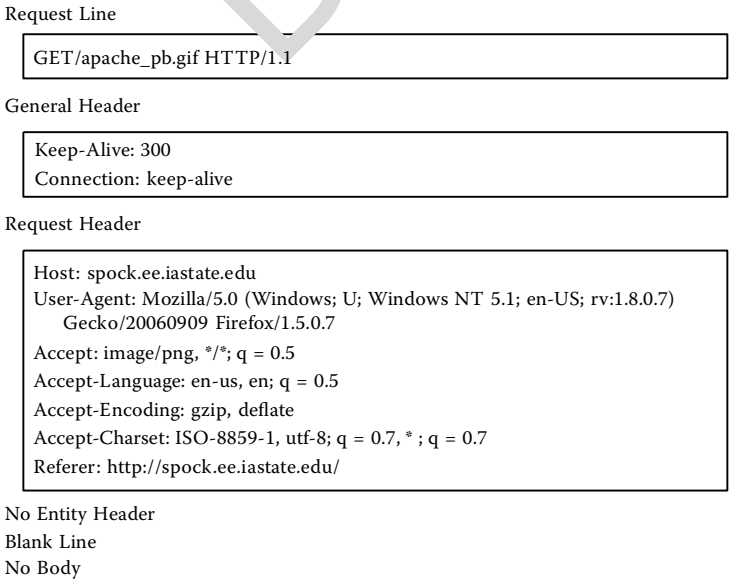


Figure 10.10: HTTP request message.

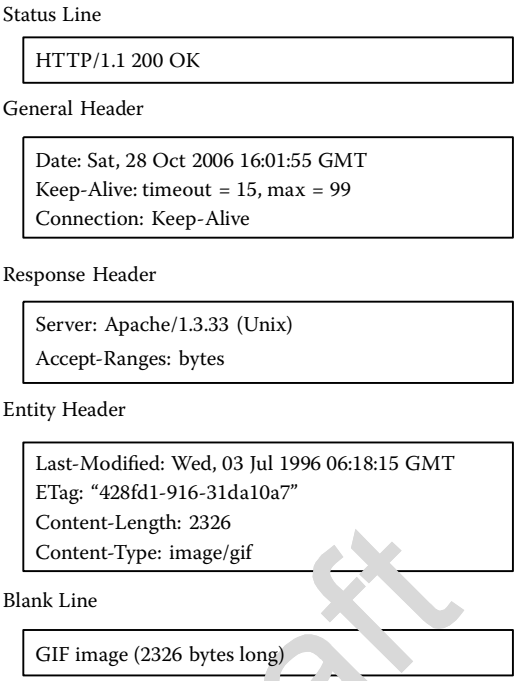


Figure 10.11: HTTP response message.

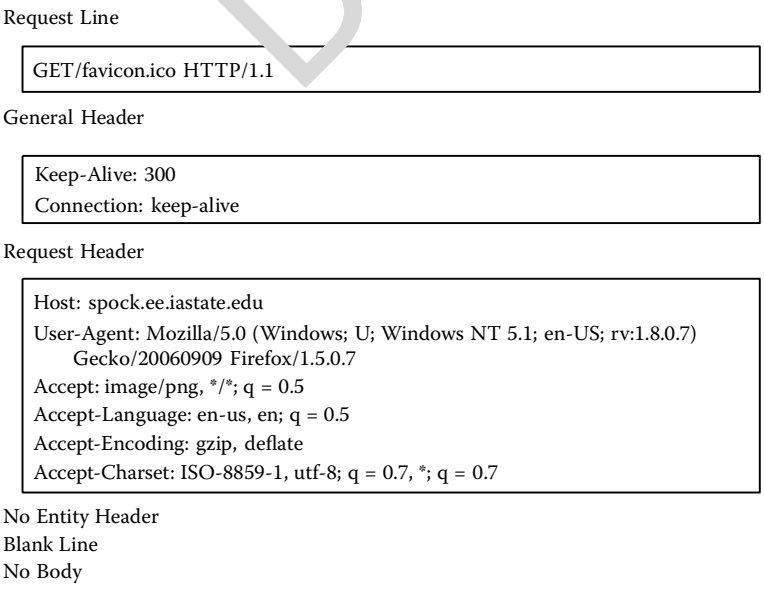


Figure 10.12: HTTP request message.

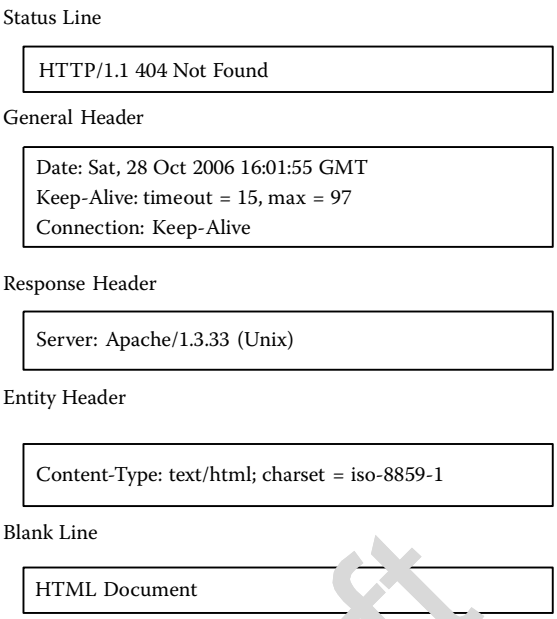


Figure 10.13: HTTP response message.

10.1.4 Vulnerabilities, Attacks, and Countermeasures

Even though the HTTP protocol is straightforward, there are several vulnerabilities. If we refer back to the four categories of vulnerabilities and attacks discussed in Part I, we will see that authentication-based and traffic-based attacks are most common.

10.1.4.1 Header-Based Attacks

Header-based attacks are not very common since the headers are simple and any command or response that is invalid is ignored. The HTTP protocol does present some interesting problems since both the client (the browser) and the server use freeform headers that can contain several options that control the way the data is interpreted. The early versions of web servers and browsers were subject to buffer overflow attacks. There have been several attacks that were able to exploit the fixed-size buffer by sending commands that were too long. Today the bigger problem is with client- and server-side executables, which will be discussed later. As we saw in Figure 10.3, the web server can be a front end for another program by passing data directly from the browser to another application. This allows an attacker to use the HTTP protocol to transport attacks to another application.

Another header attack uses the HTTP protocol to fetch files that are not part of any set of hyperlinked documents. An attacker can search a web site for files that are sometimes left on a server by default by including a file name in the URL. These files often contain no useful information for the attacker, but sometimes the files might contain authentication information or other critical data. A common configuration error is to leave the web password file inside the document tree. If an attacker can find the file, he or she would be able to use public domain attack software to discover valid usernames and passwords.

10.1.4.2 Protocol-Based Attacks

The HTTP protocol is simple and there are very few protocol-based attacks.

10.1.4.3 Authentication-Based Attacks

Authentication-based attacks are the most common type of HTTP attacks. There are several authentication methods used in a web server, and many are implemented by the application and not directly supported by the HTTP protocol. The authentication data is sent as payload in the HTTP packets. The HTTP protocol supports authentication to control access to files stored on the web server. A web server contains documents that are stored in files that can be organized into a series of directories and subdirectories. Figure 10.14 shows a typical organizational structure for a web site where the document root contains documents and directories.

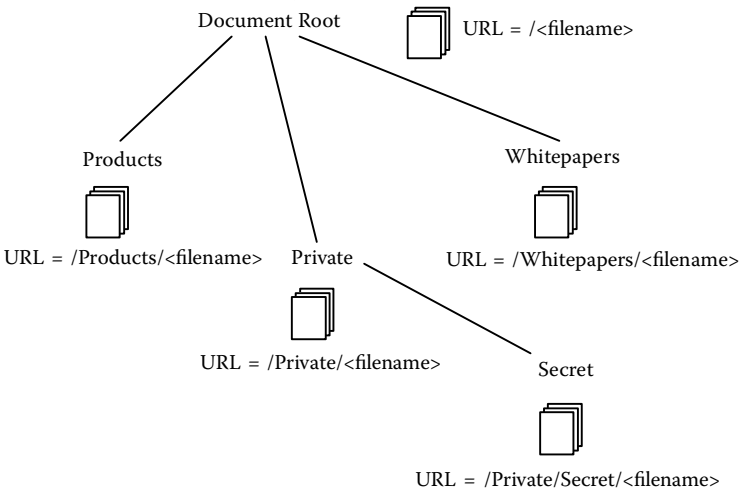


Figure 10.14: Typical web site directory structure.

Documents in the document root are addressed using a "/" followed by the file name, and documents in the subdirectories are addressed using a "/" followed by the subdirectory name(s) followed by a "/" and the file name. For example, files in the subdirectory "Products" are referred to with the URL Host://Products/<filename>.

HTTP-based authentication is designed to control access to a directory based on a username and password. In addition, access to documents in a directory can be based on the IP address of the browser. IP-based authentication was discussed in Chapter 6. For example, in Figure 10.14 the directory "Private" can be set up to require a username and password before any documents in that subdirectory can be accessed. All subdirectories of the directory "Private" are also protected by the same authentication. This is done by setting up the web server to support authentication and by placing an authentication file into the directory "Private." When the browser requests a file contained in the directory for the first time, or any subdirectory of the protected directory, the server asks the browser for authentication. For example, if a user tried to access the URL Host://Private/Secret/<filename>, he or she would be asked to provide authentication. Figure 10.15 shows the partial HTTP headers used when the browser requests a document from a protected directory.

Figure 10.15 shows a request for a document with the URL /~dougj/private/doc.html in the first request header. The server responds with a 401 response header,

Request Header

```
GET/~dougj/private/doc.html HTTP/1.1
Host: spock.ee.iastate.edu
```

Response Header

```
HTTP/1.1 401 Authorization Required
Date: Tue, 14 Nov 2006 22:37:47 GMT
Server: Apache/1.3.33 (Unix)
WWW-Authenticate: Basic realm = "Enter Password"
```

Request Header

```
GET/~dougj/private/doc.html HTTP/1.1
Host: spock.ee.iastate.edu
Authorization: Basic bG9yaWVuOmZpcnN0b25l
```

Figure 10.15: Authentication protocol exchange.

which indicates the server requires authentication. The server also sends the type of authentication and an authentication message called the realm. The realm is used to help the browser keep track of different authenticators used on the same web site. For example, another directory might have a different username and password to gain access. In this example the message is "Enter Password." The browser is responsible for prompting the user for a username and password. Once the user has entered the username and password, the browser will send another request to the server with the authenticator. The authenticator is the username and password separated by a ":" and is encoded in base64. The authenticator is included in every request to the same realm.

From a network security standpoint, the HTTP authenticator is not very secure since the username and password are sent in clear text. In addition, the passwords can be guessed and are subject to the same problem as any network-based login mechanism. The solution to the password problem is educating the user to chose secure passwords. The solution to traffic sniffing is discussed in the next section.

Another authentication-based attack is spoofing. This is where a user is tricked into going to a web site he or she believes belongs to a certain organization when in reality the site is fake. Since it is very easy to capture everything from a web site, it is easy to set up a fake web site that looks just like the real site. (There are several programs designed to download all content from a web site.) The spoofed web site can cause the user to unknowingly reveal data like passwords, account information, and personal information. Email messages that contain hyperlinks are often used as one method to get users to go to the spoofed web site. This attack requires user education to mitigate since there is no host authentication for most web sites. Encryption can help provide host authentication; however, most people do not pay much attention to whether they are connected to an encrypted web. Plus, if you normally transact business with a secure site and are tricked into going to an unencrypted spoofed site, the browser will not warn you.

10.1.4.4 Traffic-Based Attacks

A web server is vulnerable to several traffic-based attacks. Attackers can overwhelm a web server by making a large number of requests. A web server is limited in the number of simultaneous connections it will allow. Sometimes just normal traffic can cause a web server to reach its limit and start rejecting connections. This can happen when a site becomes very popular in a short period of time. There are also attack tools that will cause the same effect. It is very simple to open up multiple connections and keep them alive for the server limit. This will

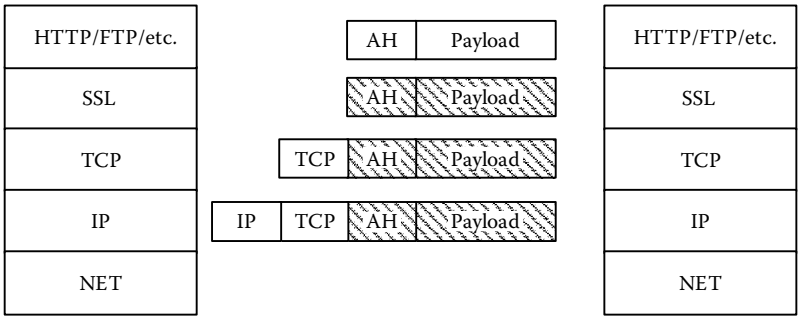


Figure 10.16: HTTPS architecture.

effectively shut down the web site. There is no good method to prevent this from happening since more often than not the goal of a web site is to attract traffic, not restrict it. Another side effect of this type of attack is to cause the router(s) close to the web server to overload with traffic and become the bottleneck. The result can be an almost complete shutdown of Internet access.

Another major problem with the HTTP protocol is that it is a clear text protocol, and therefore is subject to packet sniffing. In some cases this may be more of a privacy issue than a security issue. For example, it would be possible to tell which web site someone visited and which pages he viewed by sniffing the traffic. However, even if the web traffic is encrypted, a sniffer could still tell which IP addresses were visited. There are also cases where the web traffic is sensitive. For example, access to a bank account in clear text could reveal financial data. The primary mitigation method for network sniffing is to use encryption of the data. The primary method for encrypting a connection to a web site is HTTPS, which uses Secure Sockets Layer (SSL). SSL is used by multiple applications to provide an encrypted channel. Figure 10.16 illustrates the HTTPS architecture.

HTTPS uses port 443 and is typically transparent to the user. A web browser will indicate when the connection is encrypted through an icon (often a padlock) displayed on the screen. The browser has the public key of the web server or the public key of a signature authority that has signed a public key of a web server. In this chapter we will briefly discuss public key handling as it applies to the World Wide Web. We discussed the SSL protocol exchange in Chapter 7.

See Appendix A for information about public key encryption and how it uses certificates as a method to both authenticate and distribute the public keys.

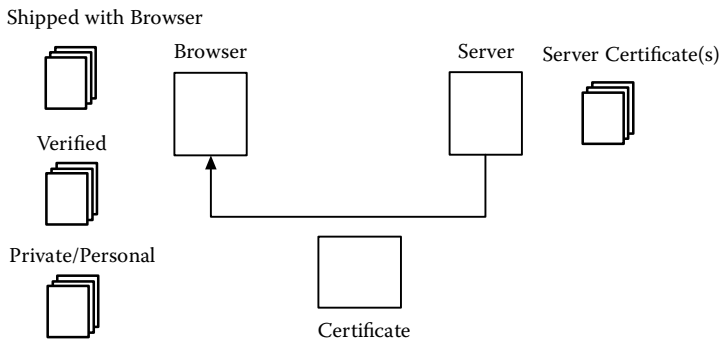


Figure 10.17: WWW certificates.

Figure 10.17 shows a web server with one or more public key certificates and a browser with multiple public key certificates.

The server presents a public key certificate to the browser to authenticate the server and to start the negotiation of a session encryption key. The server's public key certificate can be signed by a signature authority. The signature authority is used to verify the authenticity of the public key certificate. The browser examines the certificate provided by the server for the signature authority. The browser then checks its current certificates to see if it has a certificate that matches the signature authority certificate. If there is a match, the browser will use the public key of the signature authority to verify the authenticity of the received certificate. How does the browser get the first certificate, and how does it know it is valid? Companies that operate as signature authorities pay browser companies to include their certificates with the browser distribution. As shown in Figure 10.18, the certificate chain of authority is typically traced back to one of the certificates provided with the browser.

As the user visits various secure web sites, she will pick up new certificates. The user can also pick up new signature authority certificates that provide authority other certificates. A server can also provide the browser with a certificate that has not been signed by a certificate authority, or by a certificate authority known to the browser. In this case, the browser will prompt the user to see if she will accept the certificate. The user can either reject the certificate, accept it for the session, or accept it permanently.

Figure 10.18 shows that the certificate received by the browser was signed by the verified certificate. That certificate is verified by a certificate that was supplied with the browser. The received certificate is verified by taking the public

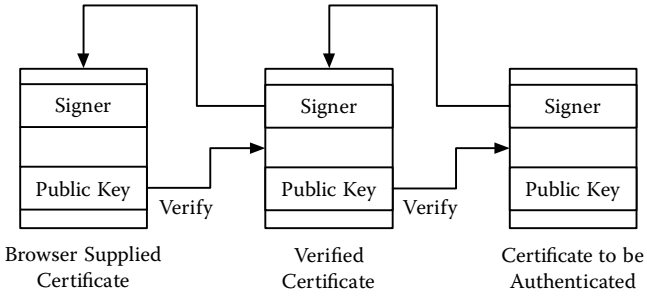


Figure 10.18: Certificate chain of authority.

key from a verified certificate and using that key to decrypt the digital signature of the certificate. Since the digital signature was created using the private key of the signature authority, we can assume the certificate was created by the signature authority. What we get from the certificate is a valid public key that has been mapped back to the server. We can use that public key to encrypt a message that only the web server should be able to decipher, since only the web server should know the private key that is associated with the public key. So in reality, the signature authority does not validate the web server; it validates that the public key in the certificate belongs to the web server. If the web server has its private key compromised, then someone could spoof a secure web site since the certificates are public knowledge. Once the browser has the public key of the web server, it can negotiate a one-time session key and use that to encrypt all traffic between the browser and the server.

Definitions

HTTPS.
An encrypted version of the HTTP protocol that uses Secure Sockets Layer (SSL).

Hyperlink.
A URL that is embedded inside a web document.

Hypertext Transfer Protocol (HTTP).
The protocol used by the World Wide Web to transfer data to and from the web servers.

Uniform Resource Locator (URL).
The address of a document in the World Wide Web.

10.2 Hypertext Markup Language (HTML)

As we discussed earlier in this chapter, the World Wide Web consists of transferring files from servers to browsers to be processed. The primary language used to display content is called Hypertext Markup Language (HTML) [12–17]. HTML documents are interpreted by the browser, and their commands dictate how the document contents are to be displayed. Since documents are processed by browsers, a server can create documents that could pose a security risk. It is not the goal of this book to examine the HTML protocol in detail; however, there are certain aspects of the protocol that can cause security problems. Figure 10.19 shows the general format of an HTML document, which consists of two parts: head and body. The head contains information used by the browser, and the body contains information to be displayed on the page. The entire HTML document consists of tags that are used to instruct the browser how to display the contents. As shown in Figure 10.19, each HTML tag has a starting marker and an ending marker.

Table 10.8 shows several of the HTML tags that can cause security problems. From a security standpoint, the three HTML commands of greastest concern are <a URL>, which allows hyperlinks; <img>, which displays images; and <APPLET>, which downloads executable code. Each of these HTML tags is briefly described next. Their security threats will be discussed in more detail in the section on HTML vulnerabilities.

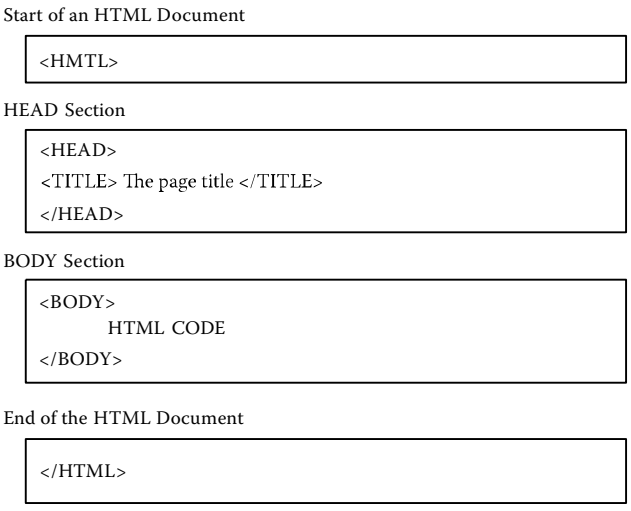


Figure 10.19: HTML format.

TABLE 10.8: Common HTML Tags

Tag	Function
<HMTL>	Tells the browser where the HTML document starts and ends
<HEAD>	Indicates the start of the head section
<TITLE>	Text to be displayed in the title bar of the browser
<BODY>	Indicates the start of the body of the document
<a URL>	Hyperlink
<img>	Embedded image to be displayed
<APPLET>	Client-side executable
<! Comment>	Used to add comments to the document

The hyperlink tag <a URL> is used to display a hyperlink on the screen. The tag has two parts: the hyperlink itself, which is a URL that points to a new document, and the text that is displayed on the screen. For example, the HTML tag shown below is a link to the document index.html on the server www.iseage.org. The browser will display the text string "Click Here" as the hyperlink.

```
<a href=http://www.iseage.org/index.html>Click Here </a>
```

The image tag is used to display an image on the screen. The tag specifies the location of the image, which can be local to the server or can reside on a different server. Unlike the hyperlink tag, which requires user action, the image tag causes the image to be downloaded and displayed by the browser. In addition to the image location, the tag also has text that will be displayed if the image cannot be loaded. The format of the image tag is shown below. The first example shows an image loaded from the local server, and the second example shows an image being loaded from the web server www.iseage.org.

```
<img src=image.gif alt="image" />
<img src="http://www.iseage.org/image.gif" alt="remote image" />
```

The APPLET tag is used to download a Java applet to the browser. The Java applet is an executable program that is run by the browser and can have parameters that are passed to the program. Applets are treated like an image in that they are displayed on the screen at a certain location and a given size as defined by the HTML code. Just like the image tag, the applets can be automatically downloaded without user intervention if the user has enabled Java applets in the

browser settings. The format of the applet tag is shown below and shows the browser calling an applet named TheApplet and passing the string "Hello Doug" as a parameter. The PARAM tag is optional and is only used if the applet needs parameters. The HTML tag <P>Hello Doug</P> is displayed if the applet cannot be executed.

```
<APPLET CODE="TheApplet.code" WIDTH=200 HEIGHT=100>
<PARAM NAME=TEXT VALUE="Hello Doug">
<P>Hello Doug</P>
</APPLET>
```

Figure 10.20 shows the HTML code to produce the web page shown in Figure 10.6. A few things to notice about the document are the comments, the hyperlink, and the embedded image.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title>Test Page for Apache Installation</title>
</head>
<!-- Background white, links blue (unvisited), navy (visited), red (active)-->
 <body bgcolor = "#FFFFFF" text = "#000000" link = "#0000FF" vlink = "#000080"
alink = "#FF0000">
 <p>If you can see this, it means that the installation of the
<a href = "http://www.apache.org/foundation/preFAQ.html">Apache web server</a>
 software on this system was successful. You may now add content to this directory and
replace this page.</p>

<hr width = "50%" size = "8" />
<h2 align = "center">Seeing this instead of the website you expected?</h2>

<p>This page is here because the site administrator has changed the configuration of this
 web server. Please <strong>contact the person responsible for maintaining this server with
 questions.</strong> The Apache Software Foundation, which wrote the web server software
this site administrator is  using, has nothing to do with maintaining this site and cannot help
resolve configuration issues.</p>

<hr width = "50%" size = "8" />
 <p>The Apache <a href = "manual/">documentation</a> has been included with this
distribution.</p>

<p>You are free to use the image below on an Apache-powered web server. Thanks for
using Apache!</p>

<div align = "center"><img src = "apache_pb.gif" alt = "" /></div>
</body>
</html>
```

Figure 10.20: Example HTML document.

10.2.1 Vulnerabilities, Attacks, and Countermeasures

10.2.1.1 Header-Based Attacks

HTML documents have a complex freeform header format. Most of the attacks against the HTML headers involve three tags: image, applet, and hyperlink. The hyperlink tag vulnerability is due to the ability to set the link information displayed on the screen to anything with no correlation to the URL. This can be used to redirect people to spoofed web sites. We often see this in HTML email, where the hyperlink text may indicate the URL is a bank, but in reality it is someplace else. This same thing can happen in a web site where a hyperlink can mislead the user and cause him to go someplace he was not expecting. Hyperlinks do not just point to other websites or other HTML documents, they can also point to other types of documents that either can contain malicious code or are themselves malicious. For example, a hyperlink may indicate the document is a certain type (HTML, text, etc.) when in fact it maybe an executable. Even though the browser asks what to do with an executable file type, the user only needs to click yes and the file will be executed. This problem will also be discussed in the section on client-side executables.

The image tag allows a web site to include images that might be stored on another web site. While this may not represent a clear security vulnerability, it can lead to privacy concerns. A web server can log the site that contained the link to the image as well as when the image was accessed. This information can enable the web server to track where a user has visited. This is called a "web bug" or a "clear gif." The image is often a 1-pixel-wide clear image that is obtained from another site. There are other methods used to track web and network activities that will be discussed in the chapter on peer-to-peer and anonymous networking.

The applet tag lets the web server download code to your browser that is run locally. The applet can also return data to the web server. There were several vulnerabilities when Java applets were first implemented that primarily dealt with the applets' ability to read files and data that should not have been accessed. This allowed attackers to write malicious code that could extract data from the computer running the browser. Today the browsers have limited what data Java applets can access. However, many users and organizations simply disable applets from untrusted sources.

The only real countermeasure for HTML header-based attacks is user education. As with most attacks that target the user through social engineering, the only way to stop them is to create a better-educated user. The biggest problem with HTML attacks is the ease with which a web site can be spoofed and the speed with

which the social engineering methods can change. We need to teach users how to handle these attacks in general, as opposed to focusing on specific examples.

10.2.1.2 Protocol-Based Attacks

HTML is not a protocol in the sense of needing a message exchange to function. In some sense we can classify many of the header-based attacks as protocol based since we are using the HTML protocol. One protocol-based attack is when the HTML code designer embeds information within the HTML document, either in the form of comments or as fixed values to be passed to a server-side executable. Since the HTML code is in clear text when processed by the browser (even if it was transferred via an encrypted channel), anyone that can display the page pointed to by the URL can read the source code. There have been cases where attackers have modified web pages and loaded them locally and used the web pages to send bogus information to the web server. The most famous cases were where attackers used modified web pages that had the price of merchandise embedded in the HMTL document. By changing the price in the HTML document, the attackers were able to purchase products at a greatly reduced price.

10.2.1.3 Authentication-Based Attacks

HTML does not directly support authentication, and therefore there are no user authentication-based vulnerabilities. The only authentication issue is host-to-user authentication since most web sites are not authenticated. This makes web site spoofing possible. The certificate-based authentication discussed earlier can help with web server authentication, but since many web sites are not encrypted, this will not solve the problem.

10.2.1.4 Traffic-Based Attacks

The only traffic-based vulnerability is traffic sniffing, which was discussed earlier along with possible countermeasures.

Definitions

HTML APPLET tag.
An instruction in HTML that allows for the insertion of a Java applet in the document.

HTML img tag.
An instruction in HTML that allows for the insertion of an image in the document.

HTML URL tag.
An instruction in HTML that allows for the insertion of a URL in the document.
Hypertext Markup Language (HTML).
A specification for the format of the documents used in the World Wide Web.
Web bug/clear gif.
A small image embedded in a web page that is used to tell when the web page is viewed.

10.3 Server-Side Security

As was shown in Figure 10.3, a web server can obtain data from executable programs. A URL can point to a program that is run on the server. Therefore, an HTML document can cause code to be executed on the server [18–27]. The most common method is using what is called the Common Gateway Interface (CGI), which is not a programming language, but a standard that defines how a program interfaces with the web server. CGI programs can be self-contained with no parameters or can be passed parameters via the URL or the HTTP POST command. The output of a CGI program is HTML code. The CGI program can also interface with other applications running on the server or on other servers. Figure 10.21 shows how a CGI program interacts with the browser, the web server, and other applications.

As we see in Figure 10.21, the browser will send the URL that is a call to a CGI script. The call can contain parameters that are passed via the URL or through the POST command. Examples of both methods are shown below. If the URL is part of a HTML document, then the server-side script will be invoked without the knowledge of the user.

```
http://www.iseage.org/cgi- bin/program.pl?name=Doug;state=IA
```

The example above shows the script program.pl being invoked with two parameters (name and state), each with a value (Doug and IA). A question mark separates the path to the executable from the parameter field, and semicolons are used to separate the parameters. This method works well when the parameters are fixed for the HTML document. However, if the user needs to enter parameters, then the FORM tag is often used in the HTML document, as shown next.

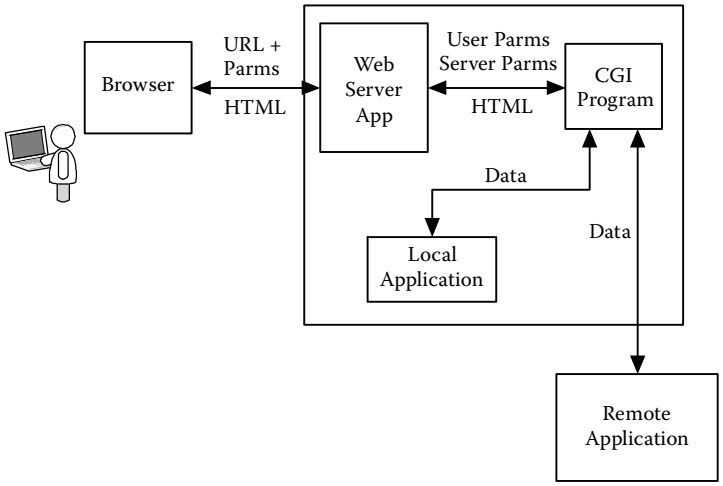


Figure 10.21: CGI interaction.

```
<FORM ACTION="/cgibin/program.pl">
Your Name: <INPUT NAME=name><BR>
Your State: <INPUT NAME=state><BR>
<INPUT TYPE=SUBMIT>
</FORM>
```


These HTML commands will produce two text input boxes labeled "Your Name:" and "Your State:" and a "Submit Query" button. When the user presses the submit query button, the values entered into the text boxes will be placed into a URL and submitted to the web server. In addition, you can use the POST method to send the data to a CGI program.

In addition to the parameters passed to the CGI script from the user, several parameters are passed to the script from the web server. Table 10.9 shows several of these parameters that are made available to the CGI script.

Once the CGI script is invoked with the parameters, the script can run with no additional interaction or it can spawn off another program. The CGI script can also communicate to a remote application like a database server. The interaction between the CGI script and other applications is not covered in the CGI specification. This interaction with other applications allows data to be passed from the browser to an application that may not be normally accessible to the browser or to anyone via the network. For example, you could create a CGI script that would display the contents of a text file to the screen by calling a program on

TABLE 10.9: CGI Parameters

Name	Function
Query_string	The string passed to the script through the URL
Remote_address	IP address of the browser
Remote_host	Host name of the browser
Server_name	Host name of the server
Server_software	Web server software
HTTP_user_agent	Browser software
HTTP_referrer	IP address of the machine that contained the link
HTTP_accept	List of document types accepted by the browser

the server. Even if the CGI script is embedded within the HMTL document with fixed parameters, an attacker could figure out where the CGI script is located and could pass its own parameters to the script, asking for the contents of any file stored on the computer.

The last step in the process for the CGI script is to send the text back to the browser in HTML. The data the CGI script gets from other applications can be in HTML or raw text, in which case the CGI script needs to convert the data to HTML.

10.3.1 Vulnerabilities, Attacks, and Countermeasures

10.3.1.1 Header-Based Attacks

Since CGI scripts accept parameters from the network, buffer overflows are a common problem. This is compounded by the fact that CGI scripts often provide network access to applications that were not designed for network access. For example, an application may be designed to read data from the keyboard and will make an assumption that the input will be limited by the OS to a fixed size. When the CGI script invokes the application, it will pass the data it receives from the network to the application. This will bypass the size restriction that is enforced by the operating system and can cause a buffer overflow in the application. The CGI script can be subject to buffer overflow attacks since they are often written in languages that do not enforce buffer protection. Another vulnerability exposed when using a CGI script to access an application is that there may not be strong type checking of the parameters by the application, and the CGI script could pass invalid parameters to the application. This happens when CGI scripts are used to access an application such as a database that may have its own application front end that checks parameters. The CGI script might bypass the front end and directly access the application.

Another common vulnerability is when an attacker uses the CGI script to access files or programs that were not intended to be accessed. This happens when there is a mistake in the CGI script and the parameters are not restricted.

When using CGI scripts, the header problems are not just with the CGI script, but also with the application, which makes mitigation very complex. CGI scripts should validate all input data and take extra care when providing access to files or other applications.

10.3.1.2 Protocol-Based Attacks

Since CGI is not really a network protocol, there are no protocol-based vulnerabilities.

10.3.1.3 Authentication-Based Attacks

The primary authentication vulnerability is not with the CGI script itself, but with the CGI script providing access to the authentication system of another application. CGI scripts can provide network access to the authentication methods of applications that were not designed to accept network-based authentication. In addition, poorly written CGI scripts may pass the user authentication as a parameter in the URL. This would make it easier for an attacker to automate password guessing. These attacks are mitigated through good design of the CGI scripts and proper use of application authentication.

Another type of authentication vulnerability is that there is no way for a client to authenticate the web server or to know if server-side executables are being used. There is very little threat to the actual browser from server-side executables, but they can be used to collect data on the user. However, CGI scripts do not provide any more of a privacy threat than any other web activity since most of the data automatically collected by a CGI script is also collected by the web server. The only additional privacy threat is if the CGI script asks for information from a user that the user should not give out, like social security numbers, etc.

10.3.1.4 Traffic-Based Attacks

There are not additional traffic-based attacks from the use of CGI scripts.

Definitions

Common Gateway Interface (CGI).
A method that defines the inputs to a server-side executable passed via the URL.

Server-side executable.
A program that runs on a web server in response to an action by a user of the web server.

10.4 Client-Side Security

In addition to HTML documents, the browsers can handle other document types, including images, word processing files, and executables [28–38]. Figure 10.22 shows the various ways a document can be handled by a browser.

As shown in Figure 10.22, the browser often needs to use a plugin, which is an executable program that enables the browser to handle various document types. Plugins are often written by third parties, many of which are individuals, rather than companies, making the software available to others. The plugins can be classified based on the type of document they handle. Executable plugins handle code that is executed on the browsers like Java applets. Document plugins handle other document types, like PDF files, and viewing plugins handle the display of other document types, like images, audio, or real-time graphics. Sometimes the document needs to be viewed by an external application like a word processor.

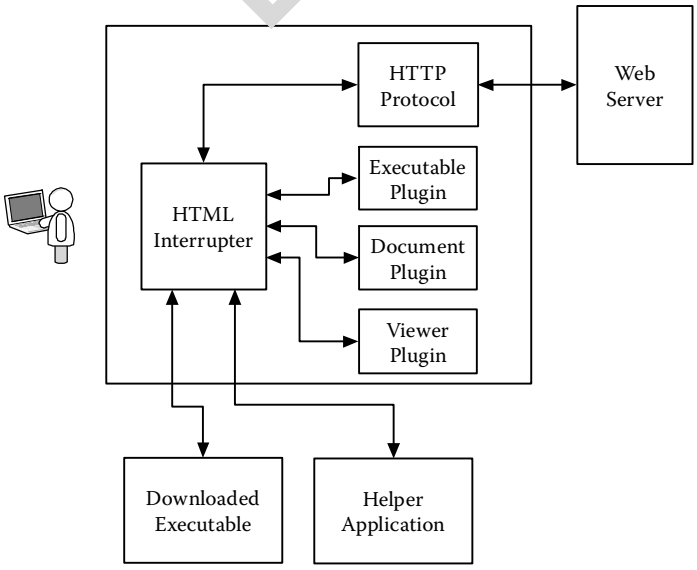


Figure 10.22: Client-side executables.

In addition to viewing various documents, the browser itself can execute code sent to it by the server. In all of the cases shown in the figure, the web server is providing documents and possibly executable code that is handled by the browser with the help of other programs. Some documents are handled automatically by the browser and are executed as part of the HTML document. The most common example of client-side executables handled as part of the HTML document is a Java applet.

Most client-side executables are downloaded and handed off to another application as the result of the user clicking on a hyperlink. In these cases, if the document is handled by a plugin, the browser will not prompt the user and will just process the document. If the document is to be opened by a helper application or is to be executed by the operating system, the browser will ask the user if he or she is sure about downloading the file. The user typically has three choices: download the document to a file, open or execute the document, or cancel the operation.

Another issue with client-side security is the use of cookies. Cookies are small data elements stored on the browser's computer by the web server. They contain data placed there by the server that can be read back by the server. Cookies are useful since HTTP is stateless, which means as you browse a site, it may involve multiple connections. Since each HTTP connection is independent, a web server has no way of connecting the user to his or her past activities. A web site that wishes to keep track of users can place a cookie on the browser's computer. The cookie often contains a user identifier that indicates who the user is. As you traverse the web site, the cookies may be used to track what you have done, either by rewriting the cookie or by maintaining a database on the web site that is indexed by the user id. Cookies represent more of a privacy issue than a security issue. One way cookies are used is to track users as they visit web sites. If a web site places an advertisement on the site, that advertisement can come from an advertisement clearinghouse, and if the user clicks on the advertisement, cookies can track which advertisements the user clicks on. The security of cookies has been debated for years, and in the early days of cookies, any web server could read any cookie placed by other web sites. Today cookies are protected so that only the web server that placed them can read them, though this is host name and IP based, so it is not completely secure. Another issue with cookies is when public computers are used to access the Internet. The cookies and other browser history will be updated. Another user can come by later and tell what sites other users have visited. This issue will be discussed in more detail in the chapter on peer-to-peer and anonymous Internet.

10.4.1 Vulnerabilities, Attacks, and Countermeasures

10.4.1.1 Header- and Protocol-Based Attacks

There are no header- or protocol-based vulnerabilities since there is no header or protocol beyond what has been discussed.

10.4.1.2 Authentication-Based Attacks

Most client-side executables are not authenticated, which can lead to malicious code. We need to discuss each of the three types of documents separately to understand the security vulnerabilities. Referring back to Figure 10.22, we can see that malicious documents can be handled by the plugin, a helper application, or can be run on the browser's computer.

Plugin attacks can be the hardest to detect and mitigate since they are handled automatically and the user sometimes does not even know that the plugin has been invoked. In the case of document and viewer plugins there are not many vulnerabilites, but if there are, they can be difficult to mitigate, since the plugin is written by a third party. The Java executable plugin, for example, had several security vulnerabilities when it was first designed. Most involved its ability to access the computer's files and to send data to other computers on the Internet. Today Java has been locked down so that is has limited access to files and to where it can communicate. Browsers can be set to allow executable plugins to run on a per-site basis, not at all, only if the user approves, or all the time. It is common to either disable or prompt the user when a Java applet is to be activated. There are some web sites that will not function unless Java applets are allowed.

The vulnerabilities associated with helper applications are the concern of the helper application. There have been macro viruses that have attacked word processors, and the web can be a method to propagate the virus; however, email seems to be the preferred method of propagating helper application attacks.

Executable files that are downloaded through the web are the largest client-side threat. The web provides an easy method to download executable files and can contain malicious code like Trojan horses, spyware, and key loggers. The malicious code may be embedded in other applications that actually perform the functions that they are reported to do. While vulnerabilities exploited by these executables are not network based, the network enables the code to be downloaded and the web facilitates the download. Sometimes email is used to draw people to the web site that contains the malicious code. By using email to entice someone to a web site and then using social engineering to convince him to download and execute a program, the attacker can bypass email virus scanners.

The best mitigation for these attacks is user education and good client-side security techniques like local virus scanners and personal firewalls. It is difficult to deploy network-based mitigation techniques against client-side attacks since the data transfer can be encrypted and the number of possible attack methods is large.

10.4.1.3 Traffic-Based Attacks

There are not very many traffic-based vulnerabilities associated with client-side executables, other than the sniffing vulnerability, which is less of an issue on client-side downloads. There are cases when the client-side executable can generate a large amount of traffic, which can cause network problems. For example, there are plugins and web sites that provide real-time stock market information or real-time weather data. There is a large amount of traffic that can be generated by the server to the client-side plugin. User education and company policies may help mitigate the traffic volume problem. Another method is to block access to certain web sites that provide excess data. This is discussed in the next section as a general web countermeasure.

Definitions

Browser plugin.
An application that is added to a Web browser. The applications are designed to handle non-HTML data.

Cookies.
Files place on the computer running the browser by a web server. The files are used by a web server to help track the user's activity on the web site.

Helper application.
An application that is called by the browser to handle data that is not handled by the browser or a browser plugin.

Java applet.
A Java executable that is downloaded from a web server and run by the browser using the Java plugin.

10.5 General Web Countermeasures

There are two active network components to the World Wide Web: web server and browser. Each has a different set of requirements for network-based countermeasures. The server needs to be protected from attacks that can target both the

web server application and the actual server. The general network-based countermeasures for the web server are the same types of countermeasures used to protect the network from an attack and will be discussed in a later chapter.

The client side has to protect the user from attacks that are often initiated by the actions of the user. As we have seen in this chapter, the user can be directed to spoofed web sites, can be tricked into downloading malicious code, and can be enticed into going to web sites that are inappropriate. One method to help protect the user is to use web filtering to prevent the user from going to locations that can cause harm, or to prevent the user from transferring files that she should not [39–46]. Many of these filters are installed to prevent the user from going to inappropriate web sites. This is not really a security issue; however, web filters can keep users from sites that could cause security breaches from malicious code. There are two categories of web filters: the URL filter and a content filter. Both categories can be placed on a single device.

10.5.1 URL Filtering

One of the early methods of client-side web protection is the URL filter, which controls which URL a user is allowed to visit. The URL filter decision is based on the final destination or URL that is requested. There are three primary methods of URL filtering—client side, proxy, and network—and each method can implement a blacklist of restricted sites or a whitelist of the only sites that can be visited. As with any blacklist-based filter, the biggest issue is keeping the list current. This is very difficult and labor intensive. If the list is to remain current, the list provider will need to constantly search the Internet for URLs that match the filtering criteria. Typically the list provider groups URLs in the filter list into categories. The end user can then chose which categories to block. The database of URLs often only contains the hash values of the URLs and not the actual URL. This is done to speed up searching and reduce the storage requirements. It is also done to protect the intellectual property (the URL list) of the list provider.

The client-side filter is typically implemented as a software wedge that is inserted into the network protocol stack and monitors all of the web traffic. Figure 10.23 shows a possible location of the software wedge.

Placing the filter in the protocol stack makes it difficult for a user to disable. The exact location in the protocol stack is dependent on the filter implementation. As shown in the figure, the filter either has a local database of sites or has a remote database that is queried before the connection is allowed.

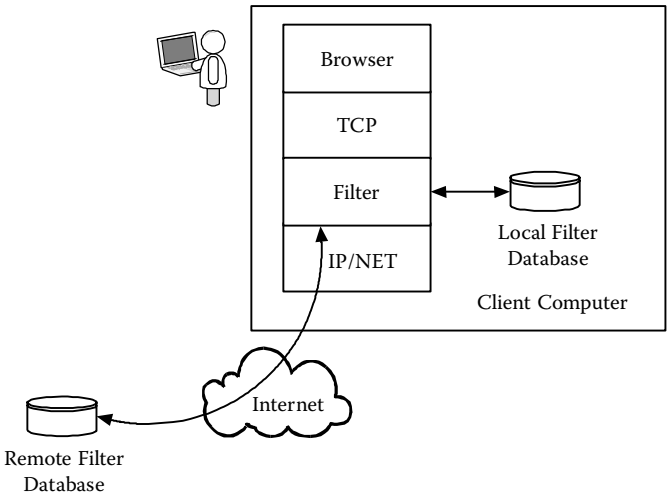


Figure 10.23: Client-side URL filter.

A proxy-based filter uses a web proxy that is a server that handles incoming web requests from the browsers and will retrieve the document from the Internet or a local cache. Web proxies were designed to reduce network traffic and speed up requests since they cache the answers to requests. In order to use a web proxy, the browser needs to be told that it should ask the proxy for the URL. Figure 10.24 shows the interactions among the browser, proxy server, and remote web site. The filter database can be housed in the proxy server or can be remote, just as with the client-side filter.

As we see in Figure 10.24, the browser sends an HTTP request to the proxy that contains the URL of the document. The IP address used is the IP address

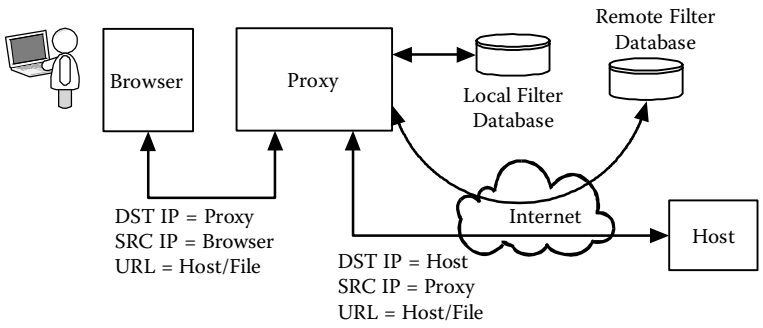


Figure 10.24: Web proxy filter.

of the proxy server. This is different from a normal HTTP request, where the IP address would be the IP address of the destination host that has the document. The proxy then sends an HTTP request to the final host, and the response is cached and returned to the browser. If the URL violates the filter rule, then the proxy can return a web page indicating the user has requested a URL that is not allowed. As we will see in the chapter on anonymous Internet, the proxy can also be used to help hide the clients since every client using the proxy will have the same return address as far as the web server is concerned.

A network-based URL filter sits at the egress point of the network and examines the traffic across the network. Unlike the client- or proxy-based approach, there is no need to change the clients. The network-based approach can be implemented as a network device like a router and is sometimes part of the firewall. In this case the device needs to have an IP address and will also route packets. The network-based filter can also be implemented as a transparent device. A transparent device does not actually route traffic; it sniffs the traffic on the network and determines if the connection should be terminated. There are two types of transparent network filters: inline and passive. The inline device has two network connections and passes all traffic from one port to another port while listening to the traffic. The passive device sniffs the traffic using a promiscuous mode interface. Figure 10.25 shows the three different network-based filters.

Each of the three methods can use a local database or a remote database. Another difference between the network-based filter and the client- or proxy-based filter is the method used to stop the connection. In the proxy- or client-based filter it is easy to stop the connection and return a blocking message. The client-side filter can return the block as network traffic (a block message as part of an HTML document) or as a message through the operating system. As discussed above, the proxy can return an HTML document that contains the blocking message. In the case of the network-based filter, the source computer has already made a network connection with the destination, and the URL is carried as network traffic. There are two primary methods used depending on the desired outcome. If the connection is only to be terminated, then the device can send a packet to the source and destination computers telling each side to terminate the connection. The network filter spoofs the IP addresses of the browser and the server to make each side think the other side requested the connection termination.

The other method is to steal the connection from the server and send an HTTP redirect message back to the browser. This is done by sending a connection terminate packet to the server pretending to be the client, and by sending an HTTP redirect to the browser pretending to be the server. The browser is then

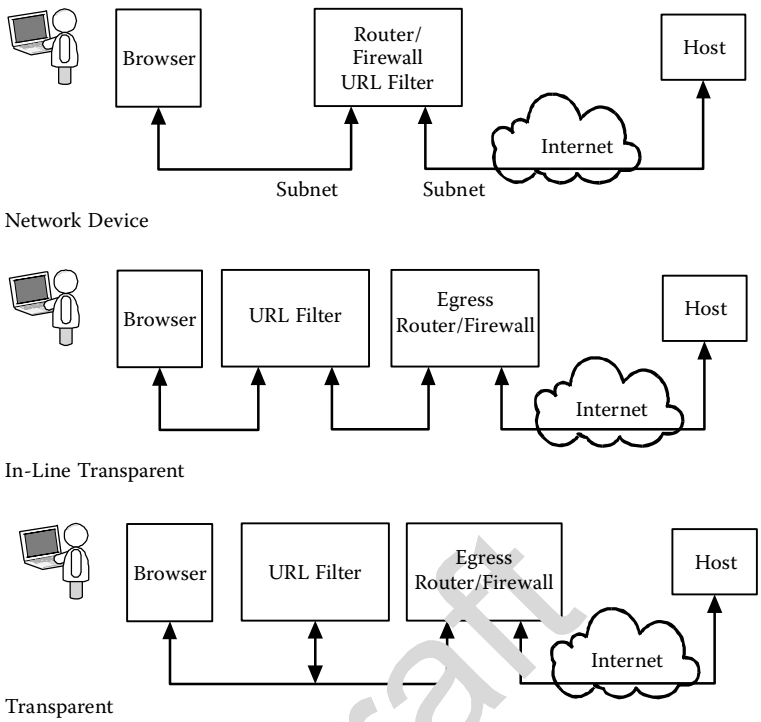


Figure 10.25: Network-based URL filter.

redirected to a web page that tells it what it did wrong. Figure 10.26 shows the two methods and the values of the IP addresses that are spoofed by the network filter.

There are proxies designed to bypass network-based filters by using SSL to encrypt the traffic between the browser and the proxy. This can be handled by blocking the IP address of the proxy. There are other methods to bypass filters and to remain anonymous on the Internet that will be discussed in Chapter 9.

10.5.2 Content Filtering

Content filtering takes the idea of URL filtering one step further and tries to examine the HTTP payload. The implementation of a content filter is no different than that of a URL filter, except they are often network devices and not client-side applications. Some URL filters also implement content filtering. There are two primary types of content filters, inbound and outbound. The only difference between the two is what type of content they are looking for.

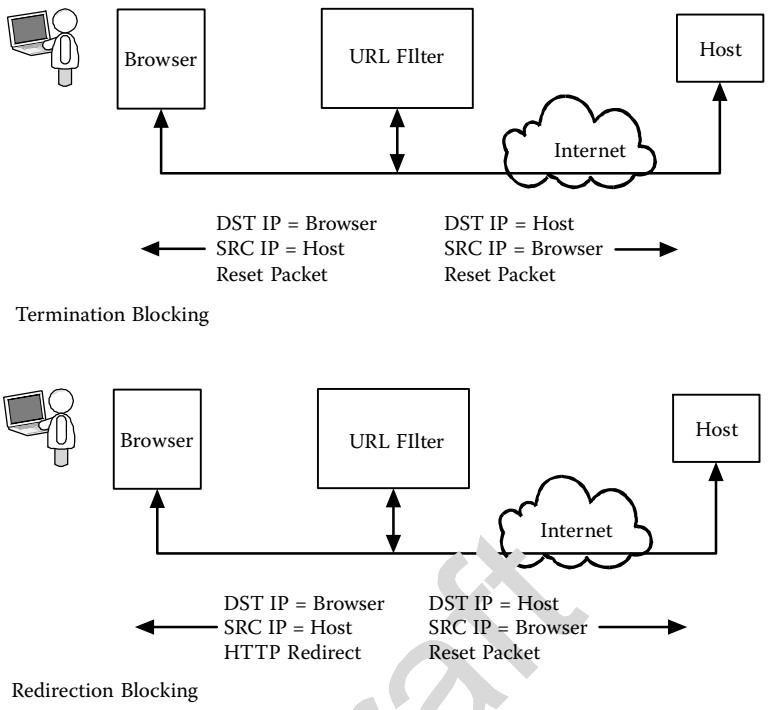


Figure 10.26: Network connection blocking.

An inbound content filter is primarily concerned with web-based attacks against the browser. There are some inbound web content filters that also focus on protecting web servers from invalid data. These devices work like virus scanners in that they look through the payload and try to determine if the payload contains malicious code. This is more difficult with the web since the traffic is close to real time. With email a virus scanner can hold the messages and then release them if there are no problems. An inbound content filter works best with a proxy server so it can store and then forward content, as shown in Figure 10.27.

Figure 10.27 shows the browser making a request for a document through the proxy. The proxy downloads the document and examines it for malicious code. If the document is clean, it is passed to the browser. If the document is not clean, the proxy either sends an empty document or sends a redirect to indicate there was a problem with the document. However, if the URL uses another method to transfer the file (like the File Transfer Protocol [FTP]), then a web filter will not stop the malicious code.

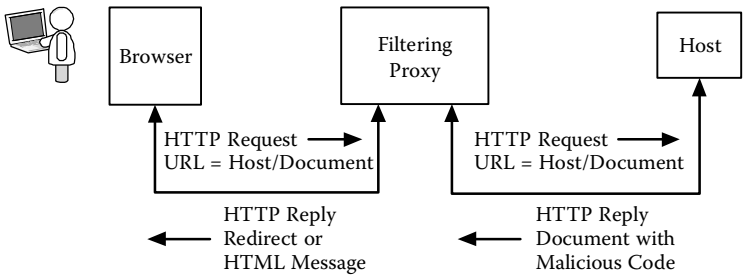


Figure 10.27: Proxy-based content filter.

An outbound content filter will examine the HTTP content for information that should not leave an organization. While this is not a direct security threat, it can be a privacy issue. An outbound content filter will often use a blacklist of content to be stopped. The filter can also use expressions to filter out content. For example, social security numbers could be matched by looking for three numbers and a dash followed by two numbers and a dash followed by four numbers. Of course, the problem with this method is how to create expressions that will stop all social security numbers and allow non–social security numbers to pass. An outbound filter is implemented the same way as a proxy.

There are several problems with content filters that make them difficult to use. First, is the issue of marking a good document as bad (called a false positive). Other content can also be a problem. At one time there was a URL filter on the market that also tried to block based on images. It examined the image and tried to determine if there were enough flesh tones to call the image pornography. This did not work very well since it was hard to determine what flesh tones are and if the flesh tones belonged to people. Another attempt at URL filtering was to filter on words or phrases in the document, like *breast*. This led to problems of banning sites talking about Thanksgiving and turkey breasts.

Another problem is missing bad documents (called false negative). There is often a trade-off between false negatives and false positives. If the filter rules are broad, they may miss very few bad documents but may stop a large number of good documents. If the rules are restrictive, then a large number of bad documents may get through.

Another problem with content filters is encrypted content. The filter is unable to open encrypted content and will not be able to determine if it should be stopped. In addition, they often have problems with compressed content.

In general, URL and content filters have a place in network security and do offer some benefits. They should also be coupled with user education and company policies. As we have seen, technology alone cannot stop a user from doing things that can cause security problems.

Homework Problems and Lab Experiments

Homework Problems

1. Show the HTTP commands to do the following:
   a. Get the web page "/index.html"
   b. Get the web page "/files/index.html"
   c. Call the CGI "/cgi-bin/print-me" with "hello there" as a parameter
2. Show the HTML code to do the following:
   a. Display a hyperlink to a local PDF file called figure.pdf.
   b. Display a hyperlink to www.dougj.net.
   c. Display a GIF image called picture.gif.
   d. Display a GIF image from a remote web site (www.dougj.net/picture.gif).
3. Show the code for a CGI UNIX shell script that prints who is logged in to the computer. Comment on whether this script should be made available to the public.
4. Research the growth of the World Wide Web, including the number of pages and the number web sites.
5. Research software that captures an entire web site and comment on how this software could be used for good and bad.
6. Research software that scans a web server for vulnerabilities.
7. Research the HTML format and software that creates HTML.
8. Research Structure Query Language (SQL) injection attacks along with other server-side injection attacks. Comment on how these attacks could be mitigated.

9. Research the various common server-side scripting languages.
10. Research client-side attacks and compare them against the taxonomy.
11. Research the Request for Comments (RFCs) associated with the HTTP protocol and the HTTPS protocol.
12. Research several URL filtering companies and determine the size of their URL filtering list and the number of filtering categories. Also determine if they are proxy based, client based, or a passive network device.

Lab Experiments

1. Use TELNET to connect to the web server in the test lab. Issue the command to get the main web page.
2. Use tcpdump or wireshark to capture a web session between a machine in the lab and a web server. Click on several links on the web server. Look at the network capture and comment on the number of connections needed and the amount and type of data transfers. Examine the headers used. Examine the source of the main page and relate it to the network caputure.
3. Use tcpdump or wireshark to capture a web session between a machine in the lab and a web server. Click on the link to the password protected and enter the password. Look at the network capture. Find and decode the password.
4. Use tcpdump or wireshark to capture a web session between a machine in the lab and a web server. Then capture a secure web session between a machine in the lab and a secure web server. Comment on what you see and the differences between the two captures.
5. Examine the contents of a browser's certificates and cookies. Comment on what information can be obtained and how it could be used for good and for bad.
6. Examine the log file of the web server in the test lab. Comment on how these files could be used to help secure the web server.

Programming Problems

1. Download the file spam.tar from ftp://www.dougj.net. Using this program as a framework, perform the following:

   a. Modify the program to ask for the header using the HTTP HEAD command. Use the parameter (-h host) to pass in the host address of the web server and the parameter (-f filename) to pass in the filename of the target file.

   b. Use the program to search for files on the web server. Test the program by checking for index.htm.

   c. Comment on how this program could be used to attack a web site.

2. Use the code you downloaded for Chapter 5 and modified in Chapters 6 and 9. Add code to perform the following:

   a. Decode and print HTTP payload. Print the payload in ASCII.

   b. Add to the set of counters a counter for the number of HTTP packets. Add code to print the values of this counter to the subroutine program_ending().

3. Write a CGI script that prints the parameters it was passed.

References

[1] Catledge, L. D., and J. E. Pitkow. 1995. Characterizing browsing strategies in the world-wide web. *Computer Networks and ISDN Systems* 27:1065–73.

[2] Lawrence, S., and C. L. Giles. 1998. Searching the World Wide Web. *Science* 280:98.

[3] Albert, R., H. Jeong, and A. L. Barabasi. 1999. The diameter of the World Wide Web. Arxiv preprint cond-mat/9907038.

[4] Vass, J., et al. 1998. The World Wide Web. *IEEE Potentials* 17:33–37.

[5] Murtaza, S. S., and H. Choong Seon. 2005. A conceptual architecture for the uniform identification of objects. Paper presented at Fourth Annual ACIS International Conference on Computer and Information Science. Jeju Island, South Korea: 100–104.

[6] Schatz, B. R., and J. B. Hardin. 1994. NCSA mosaic and the World Wide Web: Global hypermedia protocols for the internet. *Science* 265:895–901.

[7] Berners-Lee, T., et al. 1992. World-wide web: The information universe. *Internet Research* 2:52–58.

[8] Berners-Lee, T. Hypertext transfer protocol. 1996. Work in progress of the HTTP working group of the IETF.< URL: ftp://nic.merit.edu/documents/internet-drafts/draft-fielding-http-spec-00.txt.

[9] Fielding, R., et al. 1999. *Hypertext transfer protocol—http/1.1*. RFC 2616.

[10] Fielding, R., et al. 1997. *Hypertext transfer protocol—http/1.1*. RFC 2068.

[11] Touch, J., J. Heidemann, and K. Obraczka. 1998. Analysis of HTTP performance. ISI Research Report ISI/RR-98-463, USC/Information Sciences Institute. http://www.Isi.edu/touch/pubs/http-perf96.

[12] Raggett, D., A. Le Hors, and I. Jacobs. 1999. HTML 4.01 specification. Paper presented at W3C Recommendation REC-html401-19991224, World Wide Web Consortium (W3C). Cambridge, MA.

[13] Berners-Lee, T., J. Hendler, and O. Lassila. 2001. The semantic web. *Scientific American* 284:28–37.

[14] Lemay, L. 1994. *Teach yourself web publishing with HTML in a week*. Indianapolis: Sam's Publishing.

[15] Niederst, J. 2003. *Learning web design: A beginner's guide to HTML, graphics, and beyond*. O'Reilly Media.

[16] Hendler, J. 2003. Communication: Enhanced: Science and the semantic web. *Science* 299:520–21.

[17] Pfaffenberger, B., and B. Karow. 2000. *HTML 4 bible*. New York: Wiley.

[18] Kirda, E., et al. 2006. Noxes: A client-side solution for mitigating cross-site scripting attacks. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, Dijon, France: 330–37.

[19] Jiang, S., S. Smith, and K. Minami. 2001. Securing web servers against insider attack. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 2001)*, New Orleans: 265–76.

[20] Thiemann, P. 2005. An embedded domain-specific language for type-safe server-side web scripting. *ACM Transactions on Internet Technology (TOIT)* 5:1–46.

[21] Xie, Y., and A. Aiken. 2006. Static detection of security vulnerabilities in scripting languages. In *Proceedings of the 15th USENIX Security Symposium*, Vancouver, B.C., Canada: 179–92.

[22] Minamide, Y. 2005. Static approximation of dynamically generated web pages. In *Proceedings of the 14th International Conference on World Wide Web*, Chiba, Japan: 432–41.

[23] Jim, T., N. Swamy, and M. Hicks. 2007. Defeating script injection attacks with browser-enforced embedded policies. In *Proceedings of the 16th International Conference on World Wide Web*, Banff, Alberta, Canada: 601–10.

[24] Yu, D., et al. 2007. Javascript instrumentation for browser security. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Nice, France: 237–49.

[25] Erlingsson, U., B. Livshits, and Y. Xie. 2007. End-to-end web application security. Paper presented at Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS XI). San Diego, CA.

[26] Huseby, S. H. 2005. Common security problems in the code of dynamic web applications. Paper presented at Web Application Security Consortium. http://www.webappsec.org/projects/articles/062105.TX?

[27] Kumar, A., R. Chandran, and V. Vasudevan. 2006. Web application security: The next battleground. In *Enhancing Computer Security with Smart Technology*. Boca Raton, FL: CRC Press, 41–72.

[28] Marchesini, J., S. W. Smith, and M. Zhao. 2005. Keyjacking: The surprising insecurity of client-side SSL. *Computers and Security* 24:109–23.

[29] Jovanovic, N., C. Kruegel, and E. Kirda. 2006. Pixy: A static analysis tool for detecting web application vulnerabilities. Paper presented at IEEE Symposium on Security and Privacy. Oakland, CA.

[30] Jackson, C., et al. 2006. Protecting browser state from web privacy attacks. In *Proceedings of the 15th International Conference on World Wide Web*, 737–44.

[31] Kirda, E., and C. Kruegel. 2005. Protecting users against phishing attacks with antiphish. Paper presented at Proceedings of 29th COMPSAC. Edinburgh, Scotland.

[32] Raffetseder, T., E. Kirda, and C. Kruegel. 2007. Building anti-phishing browser plug-ins: An experience report. Paper presented at Proceedings of the Third International Workshop on Software Engineering for Secure Systems. Minneapolis, MN.

[33] Jakobsson, M., and S. Stamm. 2006. Invasive browser sniffing and countermeasures. In *Proceedings of the 15th International Conference on World Wide Web*, 523–32.

[34] Reynaud-Plantey, D. 2005. New threats of Java viruses. *Journal in Computer Virology* 1:32–43.

[35] Fu, S., and C. Z. Xu. 2006. Mobile code and security. In *Handbook of information security*. Hoboken, NJ: John Wiley & Sons, V. III, Chapter 144.

[36] Tilevich, E., Y. Smaragdakis, and M. Handte. 2005. Appletizing: Running legacy Java code remotely from a web browser. Paper presented at IEEE International Conference on Software Maintenance (ICSM). Budapest, Hungary.

[37] Adelsbach, A., S. Gajek, and J. Schwenk. 2005. Visual spoofing of SSL protected web sites and effective countermeasures. Paper presented at Information Security Practice and Experience Conference. Singapore.

[38] Herzog, A., and N. Shahmehri. 2005. An evaluation of Java application containers according to security requirements. In *Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05)*, Linköping, Sweden. 178–83.

[39] Kendall, K. E., and J. E. Kendall. 2002. *Systems analysis and design*. Upper Saddle River, NJ: Prentice-Hall.

[40] Bergmark, D. 2002. Collection synthesis. In *Proceedings of the 2nd ACM/IEEE-CS Joint Conference on Digital Libraries*, 253–62.

[41] Kim, J., K. Chung, and K. Choi. 2007. Spam filtering with dynamically updated URL statistics. *IEEE Security and Privacy*, 33–39.

[42] Lee, P. Y., S. C. Hui, and A. C. M. Fong. 2002. Neural networks for web content filtering. *IEEE Intelligent Systems* 17:48–57.

[43] Hammami, M., Y. Chahir, and L. Chen. 2003. Webguard: Web based adult content detection and filtering system. In *Proceedings of IEEE/WIC International Conference on Web Intelligence (WI 2003)*, Beijing, China: 574–78.

[44] Lee, P. Y., S. C. Hui, and A. C. M. Fong. 2003. A structural and content-based analysis for web filtering. *Internet Research: Electronic Networking Applications and Policy* 13:27–37.

[45] Zittrain, J., and B. Edelman. 2003. Internet filtering in China. *IEEE Internet Computing*, 70–77.

[46] Sugiyama, K., K. Hatano, and M. Yoshikawa. 2004. Adaptive web search based on user profile constructed without any effort from users. In *Proceedings of the 13th International Conference on World Wide Web*, New York, NY: 675–84.

Chapter 11

Remote Access Security

Remote access to computing resources has been a requirement of computer users since the earliest days of computing. The first computer communication systems were designed to support remote access to large mainframe computers using simple terminal devices to connect remote users using the telephone network or dedicated wires. The next phase of remote access was using personal computers as dumb terminals over the same dedicated lines or telephones lines. This led to the proliferation of terminal programs like Kermit, ProComm, Qmodem, etc. [1–3]. As desktop computers started to support networking, the requirements for remote access shifted from simple terminal access across dedicated connections to remote access from one computer to another over the Internet. This led the way to the development of several Internet protocols that supported remote terminal access from one computer to another. The first protocol developed in 1969 was called Teletype Network (TELNET). The TELNET protocol allows a computer running the TELNET client to connect to a computer running the TELNET server. Another protocol that was designed to support remote user access is called rlogin, and like TELNET, it allowed a user at one computer to connect to another computer. Unlike TELNET, rlogin was designed for UNIX-based computers and was designed for an environment where everyone was trusted. In addition to TELNET and rlogin, other protocols have been developed over the years to support remote access. Another protocol of interest is X-Windows, which allows a user on a computer running the X-Windows server to connect to a remote computer. The user's computer will display graphical content, and the user can use a mouse as an interface to the remote computer. The security vulnerabilities are the same for many of these protocols.

In addition to the need for remote terminal access there has been a need to transfer data files between computers. Several methods were developed in the early days of networking, and a couple of protocols emerged to become commonly used. The File Transfer Protocol (FTP) was one of the first to support file transfers between computers on the Internet, and like TELNET, it was designed for dissimilar computers. Another protocol that is part of the rlogin suite

of commands is called Remote Copy Protocol (RCP). RCP was designed to copy files between trusted computers running UNIX.

A new generation of file transfer protocols has emerged in the last decade that are designed for a large network of computers all sharing data with one another. The protocols are often designed to evade conventional security devices. These new networks are called peer-to-peer (P2P) networks.

In this chapter we will examine remote access protocols like TELNET, rlogin, and X-Windows. We will also examine the several file transfer protocols, including peer-to-peer protocols. After understanding these protocols and their vulnerabilities, we will look at general countermeasures and several secure protocols that are deployed today to provide secure remote access.

11.1 Terminal-Based Remote Access (TELNET, rlogin, and X-Windows)

11.1.1 TELNET

The TELNET protocol defines how a remote computer running the TELNET client can communicate with a computer running the TELNET server [4–13]. TELNET was designed so that applications on the server would not have to be modified to interact with the client TELNET application. In order to allow computers that use different character sets to communicate with each other, TELNET defines a Network Virtual Terminal (NVT) character set. Figure 11.1 shows the basic concept behind the TELNET server and its interaction with the applications.

In Figure 11.1 the TELNET server accepts connections on port 23 and connects the remote TELNET client to the applications on the server through a pseudoterminal driver. Typically the first application provides authentication and challenges the user for his or her username and password. Figure 11.1 shows the authentication application interacting with the TELNET client through the pseudoterminal and with the password file on the server. The authentication application can use any method to verify the identity of the user as long as the data is transmitted via the TELNET protocol. If the authentication fails, the authentication application will terminate the connection with the client. It is up to the authentication application to determine what constitutes an authentication failure. One example of an authentication failure is three failed login attempts.

The applications running on the server interact with the pseudoterminal driver as if a terminal was directly connected to the server. This way there are no

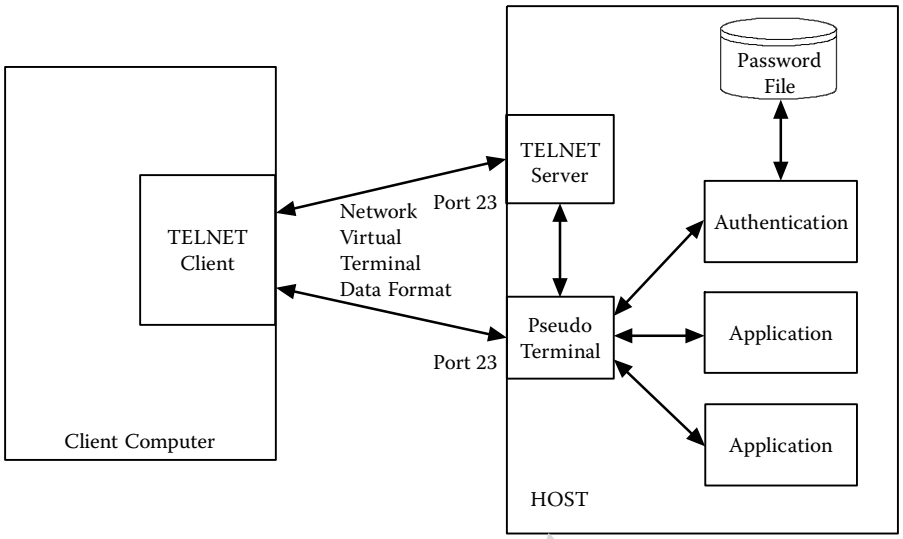


Figure 11.1: TELNET server architecture.

modifications needed in the server applications. The TELNET protocol is designed for terminal-based applications and is not suited for graphical-based systems like Microsoft Windows. There is a TELNET client that runs on MS Windows that can be used to connect to a terminal-based computer, often a UNIX system.

Figure 11.2 shows how the TELNET client and server applications interact to allow a user to connect to a remote computer.

As shown in Figure 11.2, the TELNET client application connects to the TELNET server application on port 23. The characters typed by the client are converted to NVT (which is 7-bit ASCII) and sent to the server, where they are converted to the native character set of the server and passed to the application. In order for the user to get the impression he or she is directly connected to the server application, TELNET typically sends each character typed as a separate packet using the Transmission Control Protocol (TCP) PUSH packet discussed in Chapter 7. Typically the TELNET client relies on the server side to echo every character typed by the user, which also creates the need to transfer each character as soon as possible and not wait for the TCP stream buffer to fill. In addition to transferring the user's data across the TELNET connection, TELNET uses the same connection to pass control information that is used during the initial connection, and is also used to pass special characters from the client to the server. The TELNET commands are

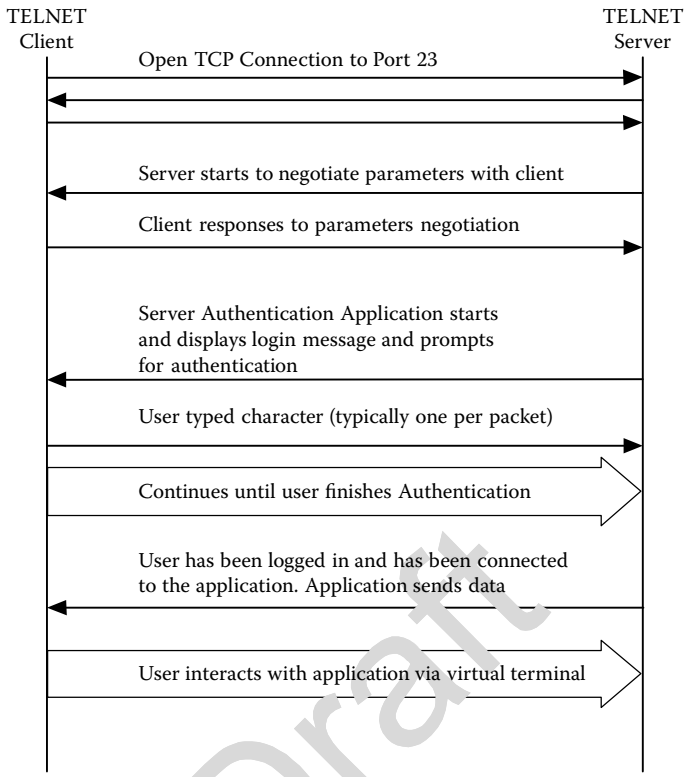


Figure 11.2: TELNET client-server interaction.

also characters, but they have the upper bit set to a 1. Table 11.1 shows some of the TELNET commands and options.

The commands BRK, IOP, and EC are used during normal data flow to handle the differences between the client's and server's interpretation of the characters. For example, the EC command allows a client that uses the control-h character as a backspace to interact with a server that uses the delete character as a backspace. When the user presses the control-h key, the TELNET client will send an EC command, which will be interrupted by the server and translated into the delete character.

In addition to the data flow commands, TELNET also uses a simple command response protocol to negotiate the initial parameters using the negotiation commands shown in the Table 11.1. By default, TELNET clients do not start the negotiation of the parameters and just establish a connection without sending any data. The TELNET server starts the parameter negotiation as shown in Table 11.2.

TABLE 11.1: TELNET Commands

Value	Abbreviation	Command
		TELNET Commands
240	SE	End of subnegotiation
241	NOP	No operation
242	DM	Data mark (stream sync character)
243	BRK	Break
244	IOP	Interrupt process
247	EC	Erase character
250	SB	Begin subnegotiation
251	WILL	Negotiation command (sender wants to enable the option)
252	WON'T	Negotiation command (sender does not want to enable the option)
253	DO	Negotiation command (sender would like the other side to enable the option)
254	DON'T	Negotiation command (sender would not like the other side to enable the option)
255	IAC	Interpret following characters as a command
		TELNET Options
ID	**RFC**	**Name**
0	856	Binary transmission
1	857	Echo
5	859	Status
24	930	Terminal type

As we see in Table 11.2, the TELNET server requests several parameters be enabled, and the client responds with what it is willing to do. Once the negotiation is complete, the TELNET server connects to the authentication application, and that application responds to the client with an authentication message. We see a single packet per character sent by the client during the authentication, and the server echos the characters back to the client. Notice that the server can respond back with multiple characters per packet when it has a longer message. Also note that the password is not echoed back to the client by the authentication application.

TABLE 11.2: TELNET Data Flow

Direction	Data	Comments
C ← S	0xff 0xfd 0x01	IAC, do echo (request client echoes)
	0xff 0xfd 0x22	IAC, do linemode (request client sends a line at a time)
	0xff 0xfb 0x05	IAC, will status (server wishes to send status info)
C → S	0xff 0xfb 0x01	IAC, will echo (client will echo characters)
	0xff 0xfc 0x22	IAC, won't linemode (client will not do linemode)
	0xff 0xfe 0x05	IAC, don't status (client does not want server to send status information)
C ← S	0xff 0xfe 0x01	IAC, don't echo (tell client not to echo)
	0xff 0xfb 0x01	IAC, will echo (tell client-server will echo)
C → S	0xff 0xfc 0x01	IAC, won't echo (tell server client will not echo)
	0xff 0xfd 0x01	IAC, do echo (tell server it is OK to echo)
C ← S	\r\n login:	Send authentication application prompt
C → S	j	First character of username
C ← S	j	Echo of the character
		Repeat until enter key is pressed
C → S	\r\n	Send carriage return + linefeed
C ← S	\r\n	Echo carriage return + linefeed
C ← S	Password:	Send authentication application prompt
C → S	p	First character of password (server will not echo)
		Repeat until enter key is pressed
C → S	\r\n	Send carriage return + linefeed
C ← S	\r\n	Echo carriage return + linefeed
C ← S		User is now connected and server application will send message

11.1.2 rlogin

TELNET was originally designed to connect dissimilar client and server computers and does not directly support any user or host authentication. TELNET simply connects the client and server applications together, and it is up to the server to authenticate the user. In 1983 a new terminal program called rlogin was included

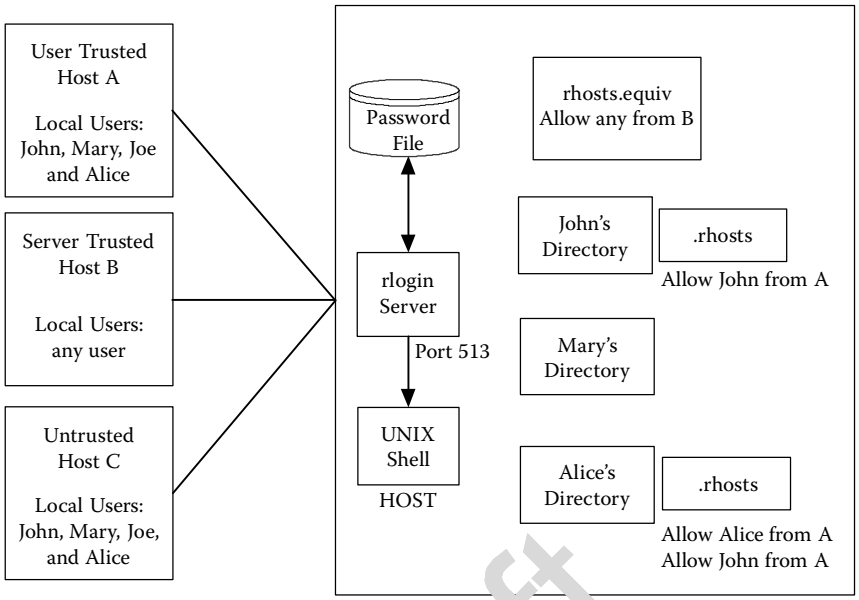


Figure 11.3: rlogin architecture.

with the release of BSD4.2 UNIX [14–20]. rlogin was designed to allow trusted UNIX machines to connect with other trusted UNIX machines with little or no user authentication. Unlike TELNET, the rlogin protocol supports authentication. The user authentication is based on the client machine IP address, the username of the client user, and the username of the server user. If these three parameters are considered trusted, then the client user is logged into the server computer without a password. If any of the parameters are untrusted, then a password is required and a prompt is displayed asking for the password. Figure 11.3 shows the architecture of the rlogin process and configuration files required for rlogin to function.

Figure 11.3 shows an rlogin server listening on port 513 waiting for a client to connect. When a client connects, it sends the local username, the remote username, and the terminal type to the server. The rlogin server consults the hosts.equiv file and the .rhosts file for the local user to determine if the remote user on the remote computer should be trusted. The hosts.equiv file can grant unauthenticated access on a systemwide scale. If the hosts.equiv file does not grant access, then the .rhosts file located in the local user's directory is checked, and if the remote user is trusted, then he or she is connected to the UNIX shell. Figure 11.4 shows a flowchart of the process used by the rlogin server to determine trust of the remote user.

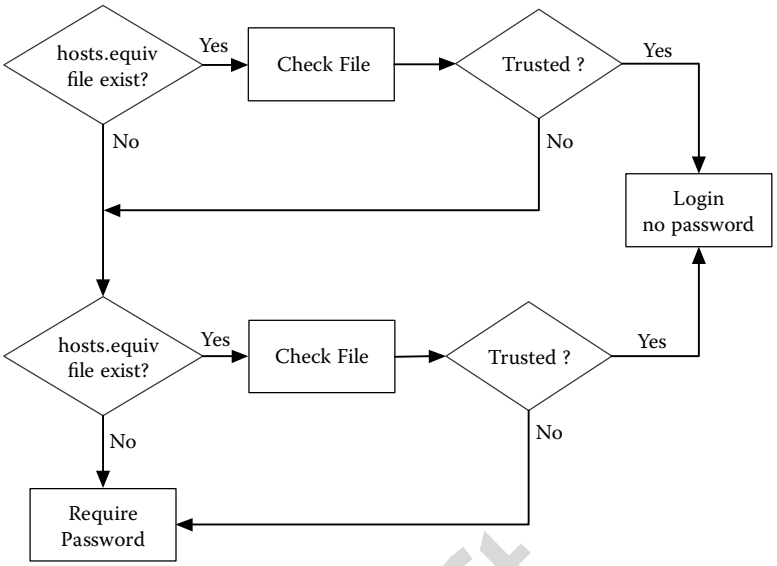


Figure 11.4: rlogin server trust flowchart.

An example of the trust mechanism is shown in Figure 11.3. The rlogin server is listening on port 513 for rlogin client connections. Of the three remote computers (A, B, C), machines A and C each have four users (John, Mary, Joe, and Alice). The hosts.equiv file is set up to allow any user from host B to log in to the server as any of the valid users on the server without authentication (a trusted user). The server has three users (John, Mary, and Alice). John and Alice each have an .rhosts file configured. The rlogin server relies on the authentication of the user's local computer to support trusted authentication to the server. In other words, the rlogin client sends the username of the user on the client as part of the protocol, and the server trusts the client is providing the username of a locally authenticated user. Table 11.3 shows the possible combinations of remote client users, remote hosts, and server users, and what the resulting trust relationship is.

If the user on a trusted client machine is trusted to log in as a user on the server, then the server will not require a password. When the user types in the rlogin command, he or she will receive a command prompt from the server. If the user on a client machine is not trusted, then the user will be prompted for a password.

As we saw in the example above, the user does not need to type any user information into the rlogin command. The rlogin client, as part of the initial message, sends the client user, the server user, and the terminal type. The terminal type is sent so the server applications know what type of terminal is running on

TABLE 11.3: rlogin Trust Example

Client Host	Client-Side User	Server-Side User	Result
	John	John	Trusted
		Mary	Not trusted
		Alice	Trusted
	Mary	John	Not trusted
		Mary	Not trusted
A		Alice	Not trusted
	Joe	John	Not trusted
		Mary	Not trusted
		Alice	Not trusted
	Alice	John	Not trusted
		Mary	Not trusted
		Alice	Trusted
B	Any user	Any user	Trusted
	John	John	Not trusted
		Mary	Not trusted
		Alice	Not trusted
	Mary	John	Not trusted
		Mary	Not trusted
C		Alice	Not trusted
	Joe	John	Not trusted
		Mary	Not trusted
		Alice	Not trusted
	Alice	John	Not trusted
		Mary	Not trusted
		Alice	Not trusted

the client. The initial protocol exchange is shown in Figure 11.5, and Table 11.4 shows the data flow for both a trusted and untrusted user.

The prompting for a password and the response are handled as part of the data flow, and not part of the initial protocol exchange. Also, the password exchange takes place in clear text.

Once the user is connected via rlogin, all characters typed by the client are sent to the server without modification, and all characters sent by the server are received by the client unaltered. Unlike TELNET, rlogin does not translate characters.

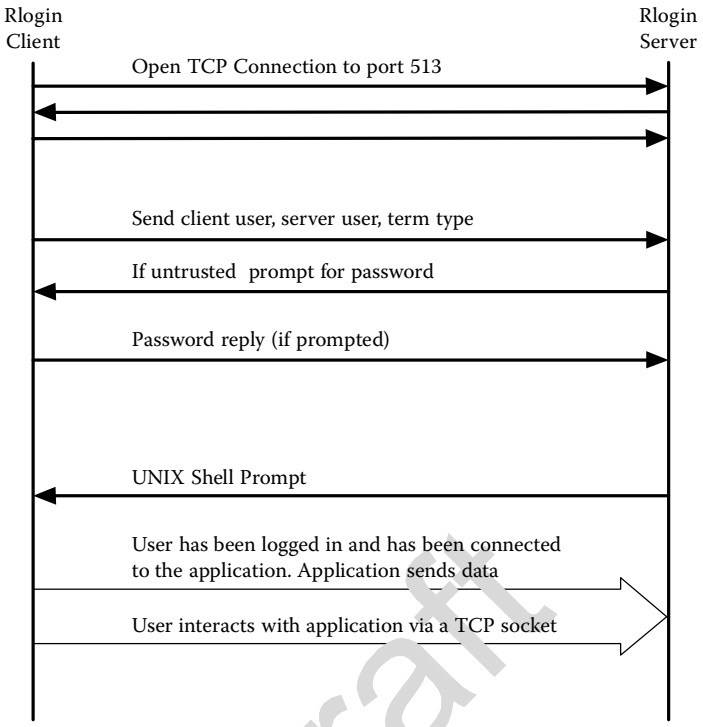


Figure 11.5: rlogin protocol exchange.

11.1.3 X-Windows

TELNET and rlogin allowed a remote user to connect to a computer as a remote terminal. This limits the types of applications a remote user can operate to command line or simple graphics applications. Both the TELNET and rlogin protocols send the terminal type to the remote server, so that the remote applications could manipulate the remote terminal. This makes it difficult to create graphical applications since the terminals vary in capabilities.

To fix this problem, a protocol called X-Windows was developed in 1984 that supported graphical applications [21–26]. The idea behind X-Windows was to define a standard set of graphical commands that can be used by an application to control a graphical display and to accept input from a keyboard and mouse. The X-Windows protocol allows application writers to create graphical user interfaces that are terminal independent. X-Windows defines a minimum set of terminal characteristics that allow for higher-quality graphical user interfaces. Figure 11.6 shows how X-Windows-based applications interface with the remote terminal.

TABLE 11.4: rlogin Data Flow

Direction	Data	Comments
C → S	john 0x00	Client-side username
	john 0x00	Server-side username
	xterm\34800 0x00	Terminal type and speed
		If authentication is required (user is untrusted)
C ← S	Password:	Prompt for password
C → S	p	First character of password (server will not echo)
		Repeat until enter key is pressed
C → S	\r	Send carriage return
C ← S	\r\n	Echo carriage return + linefeed
		If authentication worked or user was trusted
C ← S	Data from server	User is now connected and server will display the UNIX shell prompt

As shown in Figure 11.6, the terminal that provides the graphical display and mouse input is the server, and the application is the client that connects to the display. In a remote X-Windows environment the client computer often connects to the remote computer using TELNET or some other remote access protocol.

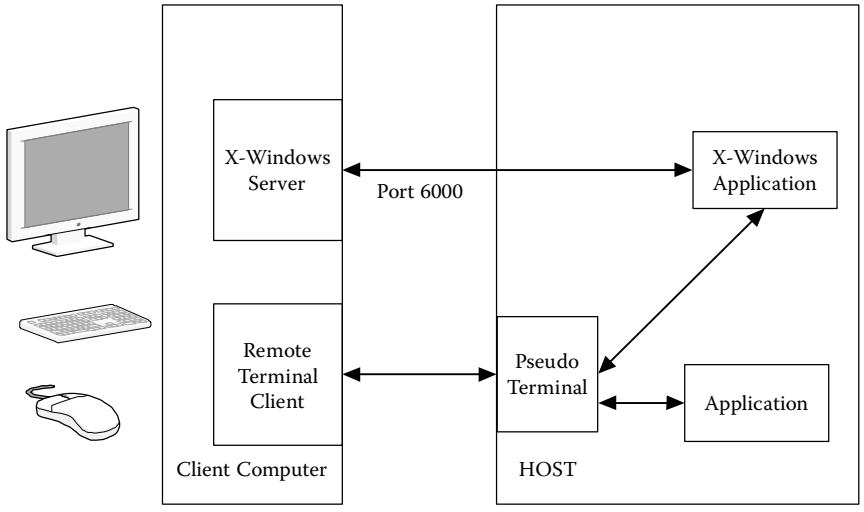


Figure 11.6: X-Windows remote architecture.

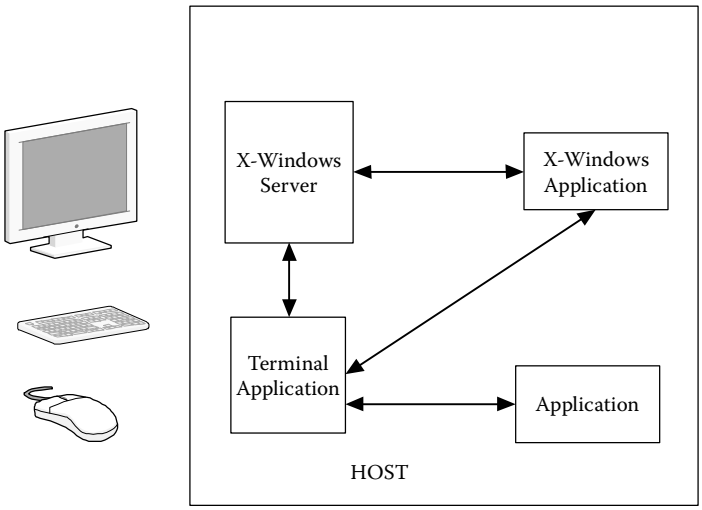


Figure 11.7: X-Windows local architecture.

Once the user has been authenticated, he or she can run an X-Windows application. That application opens a connection back to the client computer's X-Windows server. The client's X-Windows server takes graphic commands from the application and displays them on the screen, and takes input from the user via the keyboard and mouse and sends that information back to the application using the X-Windows protocol. For the purpose of this book, we will not examine the X-Windows protocol, but from a security standpoint it is important to know that it is a clear text protocol with limited host authentication.

The X-Windows terminal was originally designed as a stand-alone one, but today it is an application that can run on either the local computer or a remote computer. Figure 11.7 shows an X-Windows server local to the same host as the application.

The X-Windows protocol does not have many security vulnerabilities, and therefore we will not examine the protocol in detail. It is an open protocol with several commercial and public domain servers. Most public domain UNIX systems support X-Windows as the primary graphical user interface.

11.1.4 Vulnerabilities, Attacks, and Countermeasures

Even though the TELNET and rlogin protocols are straightforward, there are several vulnerabilities that involve authentication. X-Windows also has

authentication vulnerabilities, but is more complex since the server runs on the user's computer.

11.1.4.1 Header-Based Attacks

TELNET and rlogin do not have any headers, and therefore are not subject to header-based attacks. The X-Windows protocol does have headers and has a fairly complex encoding scheme. X-Windows is vulnerable to buffer overflow attacks. Buffer overflows can affect both the client application and the X-Windows server. Several buffer overflow vulnerabilities have been discovered in different X-Windows servers over the years.

11.1.4.2 Protocol-Based Attacks

The protocol used by TELNET and rlogin is simple and, after the initial connection, is nothing more than a direct connection from the client to the server over the TCP stream service.

TELNET does allow a user running the TELNET client access to any TCP-based application. A user could enter commands directly to an application and could create commands that violate an application layer protocol. This is not a vulnerability of the TELNET protocol and is more of a feature of the TELNET client. As we have seen in the earlier chapters of this book, we can use the TELNET client to test applications by directly connecting to them and issuing application layer commands. For this reason, we often find the TELNET client on many computers even though the use of TELNET is discouraged and is not enabled on most servers.

The X-Windows protocol is event driven and has not had many identified vulnerabilities. The X-Windows server is under control of the X-Windows application, and therefore is vulnerable to rogue applications. There is nothing built into the X-Windows protocol to help verify the integrity of the application sending commands to the server. This could also be considered a host-to-host authentication problem.

11.1.4.3 Authentication-Based Attacks

The TELNET protocol does not directly support user authentication, so it is not subject to authentication attacks. However, the TELNET server does allow a remote user to connect to the server computer and then enter a username and password. Thus, TELNET does provide remote access to the authentication mechanism of a server computer, just like Post Office Protocol version 3 (POP3)

and Internet Message Access Protocol (IMAP) allow access to a computer's authentication mechanism. This remote access can be used for password guessing. Since TELNET does not support authentication, it is up to the remote computer's authentication mechanism to protect itself from attackers. One method used is to close the TCP connection after several failed user authentication attempts on the server. This causes the TELNET client to quit and close the TCP connection. This forces an attacker to reopen the connection to continue to try passwords. The server computer's authentication mechanism will often log the failure attempts, thus providing the computer's administrator a chance to take action. Often the administrator will block the remote IP address from making new connections. In reality, TELNET is not very useful for an attacker that has to guess passwords. The best way to mitigate TELNET authentication attacks is to use something other than simple username passwords for authentication. One method is to use what is referred to as two-factor authentication. This is where the user wishing access uses two methods, like a username password and a smartcard. Since two-factor authentication is the responsibility of the application or host, it is beyond the scope of this text.

As we saw, rlogin does support authentication, and therefore is subject to authentication-based attacks. A rlogin server uses IP addresses and usernames to prove the identity of the remote user. If the remote user is untrusted, then rlogin will request a password. If a remote user is trusted, then an attacker can use rlogin to gain access from a compromised trusted host to another host in the network. An rlogin server is also vulnerable to spoofing attacks since the client provides the identity of the client-side user as part of the protocol exchange. This would allow a rogue client to place any username in the protocol message, thus pretending to be an authenticated user on the client. Figure 11.8 shows how an attacker can use rlogin to step from one computer to another.

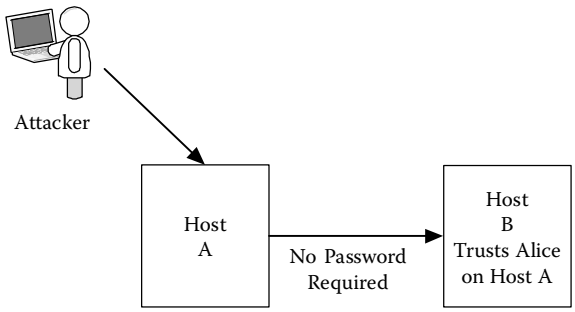


Figure 11.8: rlogin stepping attack.

As we see in Figure 11.8, the attacker gains access to host A as Alice, and since Alice on host A is trusted by host B, the attacker can gain access to host B as Alice. The attacker does not need to be authenticated as Alice on host A if it can spoof the username Alice in the rlogin protocol exchange. The best way to mitigate this is not to allow trusted hosts or users. This will require a password each time rlogin is used.

Like TELNET, rlogin will also ask for a password and is subject to password guessing attacks. This is not used very often as a method of attack since it can take too long and system administrators will be notified after several wrong guesses. The best way to mitigate these attacks is to not use the protocol or couple it with additional authentication. Since rlogin has authentication as part of the protocol, it is harder to use two-factor authentication.

The X-Windows server uses the IP address of the client to authenticate access to the server. It is possible to allow unauthenticated access to an X-Windows server. There is no user-based authentication. This means that if the trusted machine running the client code is compromised, the attacker on that machine would have access through X-Windows to the hosts running the X-Windows server.

11.1.4.4 Traffic-Based Attacks

Since TELNET, rlogin, and X-Windows are clear text protocols, they are subject to sniffing attacks. An attacker that can capture the traffic would be able to capture the username and password inside the session. An attacker would also be able to capture all of the keystrokes and responses between the client and server. This could compromise other applications and allow the attacker to obtain additional usernames and passwords that the user typed during the session. The best way to mitigate sniffing vulnerabilities is through encryption. There are a couple of methods used to provide encryption for remote access protocols. The most common is to use a protocol called Secure Shell (SSH), which replaces rlogin and provides an encrypted connection. Since SSH can protect more than just remote terminal connections, we will discuss it as part of the section on general countermeasures. There is an encrypted version of TELNET that is not used very often. It is based on a shared secret key. Since SSH is the predominant method of encrypted remote access, we will focus on that protocol as a solution.

TELNET and rlogin protocols are not susceptible to traffic flooding attacks. However, they can generate a large number of packets when the remote computer is responsible for echoing each character back to the client. Often this results in two packets for every character typed by the user. The client side is limited by

the speed at which a user can type, and therefore these protocols typically do not create a large traffic load on the network. The X-Windows protocol can produce a greater load on the network and the computing resources of the X-Windows server. Therefore, it is more susceptible to traffic-based attacks.

Definitions

Network Virtual Terminal character set.
A character set that all implementations of an application agree upon to encode their data. Typically it is 7-bit ASCII.

Stepping stone.
An attacker enters one device and from there enters another device, and continues until it reaches the target. This is done to either hide the true location of the attacker or take advantage of a trust relationship.

Trusted host.
A host that is allowed access typically based on its IP address as an identifier.

11.2 File Transfer Protocols

The need to transfer files between computers existed long before networks. The advent of networks has enabled easy transfer of files between computers. We have seen that Hypertext Transfer Protocol (HTTP) is basically a file transfer protocol used by the web browsers and servers. In this section we will explore file transfer protocols and examine their security implications.

11.2.1 File Transfer Protocol (FTP)

FTP is a common protocol used to transfer files between computers [27–32]. FTP supports user authentication and provides limited data translation between dissimilar computers. FTP also provides a common command structure to support directory and file management (create directories, list files, remove directories, etc.). The common command structure allows a user to interact with a remote computer without knowing the commands for every computer he or she wishes to transfer files between. FTP uses a simple set of ASCII commands to support the interaction between computers.

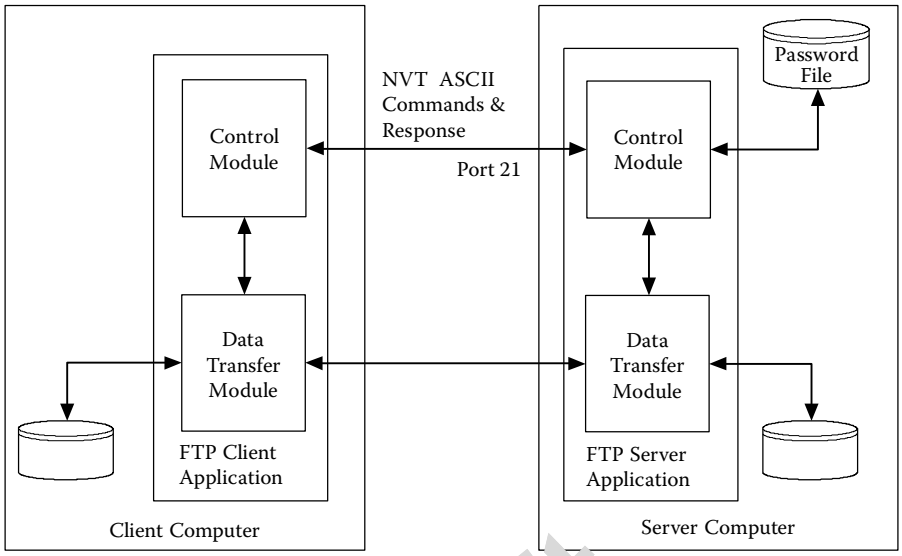


Figure 11.9: FTP client-server architecture.

To facilitate the interaction between the FTP client and the FTP server, the FTP protocol uses two TCP connections. When the FTP client connects to the server using the command port (port 21), the user is prompted for a username and password. The command–connection remains open during the session and is used to send commands and responses between the client and server. The command protocol is a command response protocol and is described below. If there is any data that needs to be transferred, the client and server will open a data connection. Depending on the configuration, the server may initiate the data connection to the client, or the client may initiate the data connection to the server. A new data connection is opened for each data transfer. Figure 11.9 shows the FTP client-server architecture and the command and data modules within the FTP client and server applications. As shown in Figure 11.9, the client and server command modules interact using the NVT ASCII that is also used by TELNET. Note that you can use a TELNET client to interact with the command module, but you will not be able to transfer any data.

As mentioned earlier, the command protocol is a simple ASCII-based command-and-response protocol that is very similar to the Simple Mail Transfer Protocol (SMTP) in that every command has a three-digit response code. Table 11.5 shows common FTP commands.

Each response code consists of a three-digit ASCII number and a text field. The first digit of the response code indicates if the command worked or failed,

TABLE 11.5: FTP Commands

Command	Action
	Authentication
USER username	Send the username to the server
PASS password	Send the user password to the server
QUIT	Finish session
	File Management
CWD directory_name	Change directory on the server
CDUP	Change to the parent directory on the server
DELE filename	Delete the file from the server
LIST directory_name	List the files on the server
MKD directory_name	Make a new directory on the server
PWD	Print the current directory on the server
RMD directory_name	Delete a directory from the server
RNFR old_file_name	Name of file on the server to be renamed
RNTO new_file_name	Name of file on the server to rename the file to
	Data Format
TYPE (A, I)	Set data transfer type, A = ASCII, I = image
	Data Port
PORT 6-digit identifier	Client sends the port number for the server to connect to for the data transfer
PASV	Server sends the port number for the client to connect to for the data transfer
	File Transfer
RETR filename(s)	Transfer the file(s) from the server to the client using the data connection
STOR filename(s)	Transfer the file(s) from the client to the server using the data connection
	Miscellaneous
HELP	Server will return information

the second digit specifies what type of code, and the third digit is used to indicate specific codes. The response code syntax is shown in Table 11.6. Table 11.7 shows the common FTP response codes.

The client and server data transfer modules use a separate connection for each data transfer. There are two methods of data transfer based on whether the client

TABLE 11.6: FTP Response Codes

Code	Response Status
1XX	Positive preliminary reply—Indicates the server will respond with another response code before the client can continue
2XX	Positive completion reply—Indicates the command was successful and a new command can be issued
3XX	Positive intermediate reply—Indicates the command was successful, but the action is held up pending receipt of another command from the client
4XX	Transient negative completion reply—Indicates the command was not accepted; however, the error is temporary
5XX	Permanent negative completion reply—Indicates the command was not accepted
Code	**Response Type**
X0X	Syntax error or unimplemented commands
X1X	Information—Reply to a request for information
X2X	Connections—Reply to a request for connection
X3X	Authentication—Reply to authentication commands
X4X	Unspecified
X5X	File system—Reply to file system-based requests

TABLE 11.7: Common FTP Response Codes

Code	Responses
150	Data connection will open
200	Command acknowledgment
220	Service ready
225	Data connection open
226	Closing data connection
230	User logged in
331	User needs password
425	Cannot open data connection
500	Syntax error
530	User login failure

or the server opens the connection. Typically we think of the server waiting for a connection from the client, which is called passive mode. The most common method used by FTP is to have the client data transfer module listen for the data connection from the server. In order for the server to know the port number, the client is waiting for the connection on the client uses the PORT command to send the IP address and port number. The server data transfer module connects to the client data transfer module and performs the data transfer specified by the client command. Either method of data transfer does add some complexity when an organization tries to use a firewall to block connections. Often the firewall blocking rules are based on port numbers. The firewall can overcome the dynamic port number assignment by trying to associate connections to open FTP command connections. If the firewall knows there is an open FTP command connection, then it can allow other connections between the FTP client and server. Figure 11.10 shows both the user interaction during an FTP session and the resulting command and data protocol interaction of the FTP session.

In Figure 11.10 the user starts by running the FTP client and connecting to the server. The data entered by the user is in bold text. The server responds with an ASCII response code (220) and a text message. The client displays the message it receives from the server. The client prompts the user for the username, and once the username has been entered, the client sends the USER command with the username typed by the user as the parameter. Note that unlike in TELNET, the username is contained in one packet. The FTP server responds with 331, indicating a password is required. The server responds with this code even if the username is not a valid user. This prevents using FTP as a method to guess valid usernames. The client prompts the user for a password and does not echo the typed password back to the user. The entered password is sent to the server using the PASS command with the user-provided password as a parameter, and the server responds with a 230 if the user was logged in successfully and returns a 530 if the authentication failed. Once the client is connected and authenticated with the server, the user can send commands to the server.

In Figure 11.10 the user requests a directory listing, which requires the client and server to open a data connection. This is shown by the client sending the port command. The first four parameters of the port command are the IP address of the client as a comma-separated list. The last two parameters are used to compute the port number the client is listening on for the server connection. In the figure these two values are 19 and 137, which are converted to a port number by multiplying 19 by 256 and adding 137. This corresponds to port number 5001. The client

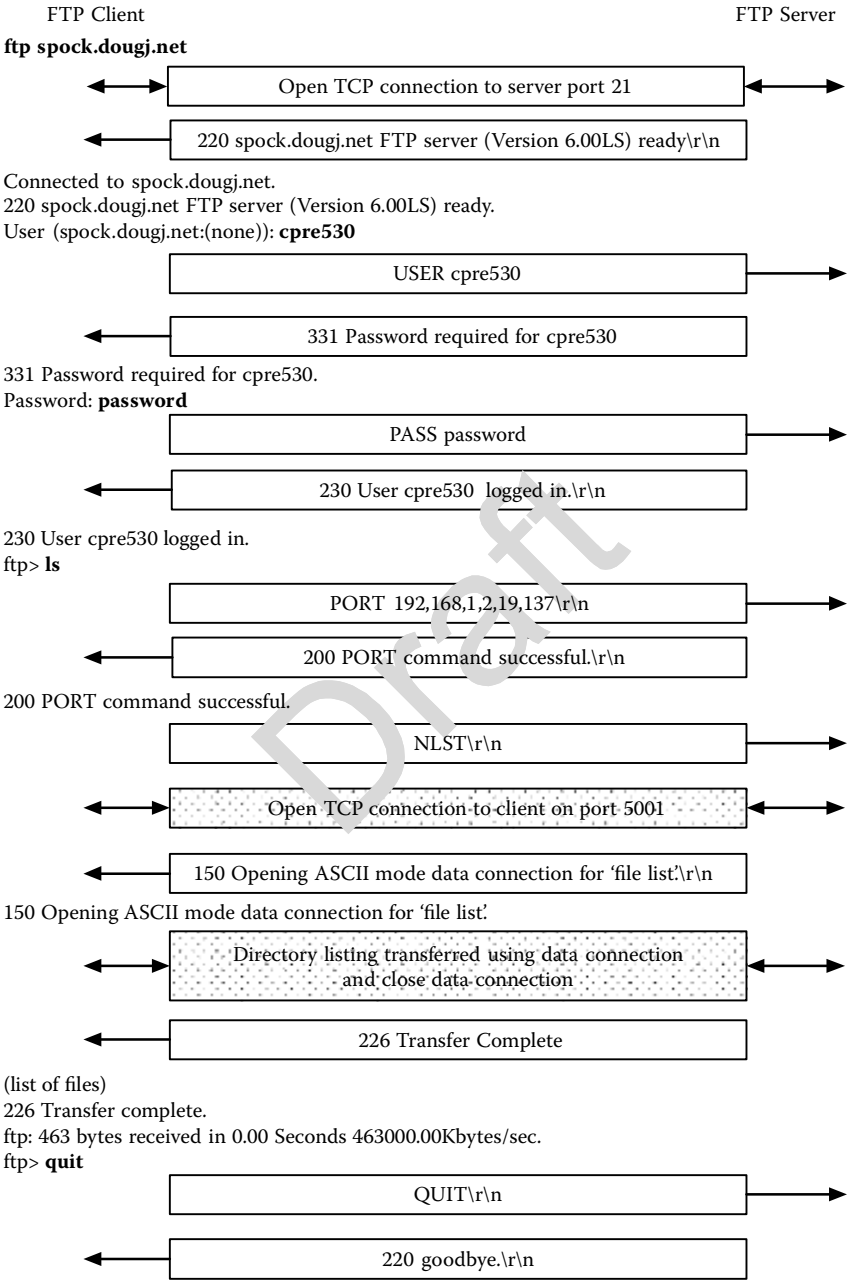


Figure 11.10: FTP client-server interaction.

then sends the NLST command, and the server opens a connection to the client and sends the data. The server also sends a response code of 150 back to the client, indicating the data connection has been opened. Once the data has been transferred, the server closes the connection and sends a response code of 226, which indicates the transfer is completed. Another data transfer starts with the client sending the port command followed by the request. The client can close the command connection by sending a QUIT command, and the server responds with a 220.

As we saw in Figure 11.10, the server requires a username and password before the client is allowed to interact with the server. In some cases it is useful to have unauthenticated file transfers. Typically an unauthenticated file transfer is limited to transfers from the server to the client. FTP is a common protocol to distribute software and other large files. In order to support unauthenticated file transfers, FTP supports what is called anonymous FTP. To use anonymous FTP, the client enters "anonymous" as the username and typically his or her email address as the password. Most servers do not check the password to see if it is a valid email address. The user can then have access to all of the files stored on the FTP site. In Figure 11.10 the only change in the protocol exchange for an anonymous file FTP session would be the username and password. The response code sent by the server after the USER command would still be 331. Table 11.8 shows a sample session with an anonymous FTP server.

Anonymous FTP is also used by web browsers when the URL is of the form ftp://machine.net/file. The only difference between user-authenticated FTP and anonymous FTP is the server configuration. Figure 11.11 shows the typical

TABLE 11.8: Anonymous FTP Access

```
$ ftp spock.dougj.net

Connected to spock.dougj.net.
220 spock.dougj.net FTP server ready.
User (spock.dougj.net:(none)):

anonymous
331 Guest login ok, type your name as password.
Password:
230 Guest login ok, access
restrictions apply.
ftp
```

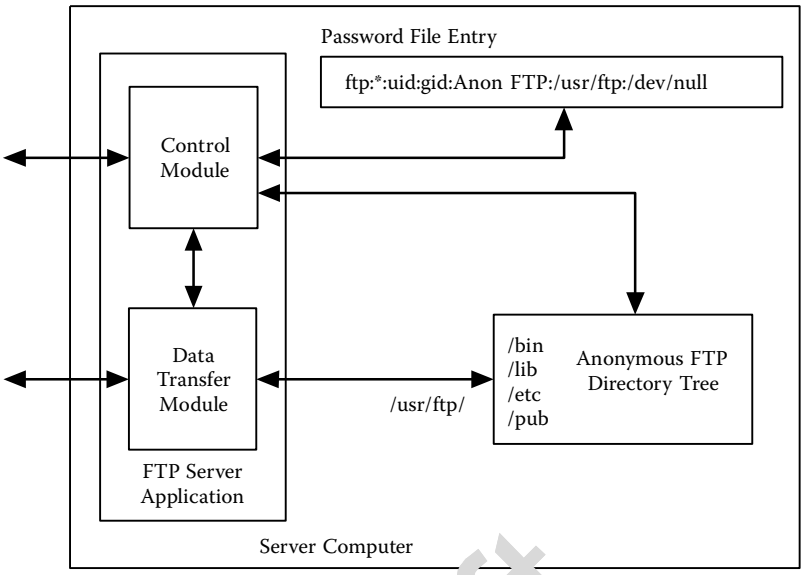


Figure 11.11: Anonymous FTP server configuration.

anonymous FTP server configuration. In this figure we see the password file has an entry for the user FTP, but there is no password for the user, nor is there any shell program. This account is disabled for all other authentication-based applications (TELNET, POP, IMAP, etc.). The home directory in this example is /usr/ftp, and this becomes the root of the directory path for the anonymous user that connects to the FTP server. This restricts the anonymous user to only the directories that are subdirectories of /usr/ftp. Since anonymous FTP uses the same FTP server, all of the commands are enabled, including the commands that allow the remote user to upload, delete, and rename files. The remote user also has access to the commands to manipulate the directories.

It should be noted that there are programs available that search the Internet for anonymous FTP sites that have writable directories. The attackers use these anonymous FTP sites to store copyrighted music, software, movies, and other illegal materials. To prevent the anonymous user from adding or changing files, the server should be configured so that all of the directories are read only.

11.2.2 Trivial FTP

There are times when we need a simple file transfer protocol that can be used to transfer files without the overhead of the TCP layer [33–38]. This is often seen in

TABLE 11.9: TFTP Packet Types

Name (opcode)	Parameters	Function
RRQ (1)	Filename (var), 0x00 Mode (var), 0x00	Read request; mode is either netascii or octet.
WRQ (2)	Filename (var), 0x00 Mode (var), 0x00	Write request; mode is either netascii or octet.
DATA (3)	Block number (2 bytes) Data (0–512 bytes)	Block number starts at 1; all blocks except the last must be 512 bytes long. A block that is less than 512 bytes is used to indicate the last block, and the file transfer is done.
ACK (4)	Block number (2 bytes)	Used to acknowledge the data block
ERROR (5)	Error number (2 bytes) Error data (var), 0x00	Used to indicate an error; the error data is text data.

diskless workstations or network devices. Trivial File Transfer Protocol (TFTP) was designed to use the User Datagram Protocol (UDP) transport layer and to operate without authentication. For the purpose of this book, we will not go into the details of this protocol other than to explore the packet format and security implications of an unauthenticated file transfer protocol. Since TFTP uses UDP, there is no guarantee of reliable and ordered data transfer. The protocol is simple and only has five packet types, as shown in Table 11.9.

The TFTP protocol is configured like anonymous FTP in that it can be confined to only allow access to part of the server's file system. Just like with anonymous FTP, care should be taken to make the directories read only. Most organizations do not need to run a TFTP server, or if they do, they only need one server.

11.2.3 RCP

Another method of file transfer, called Remote Copy Protocol (RCP), is part of the rlogin set of protocols and uses the same trust method as described in the section on rlogin. The primary difference is that RCP does not support authentication, and if the user is not trusted, then the copy will not take place.

The syntax of the command is:
rcp machine.user:source_file machine.user:destination_file

If the user and machine are trusted based on the rules described in the section on rlogin, then the copy will take place; otherwise, it will fail. There is very little reason to use RCP since it forces the use of unauthenticated access.

11.2.4 Vulnerabilities, Attacks, and Countermeasures

This section reviewed three file transfer protocols. These protocols have several common security vulnerabilities, and each has vulnerabilities that are unique to the protocol. Each protocol will be compared to the taxonomy.

11.2.4.1 Header-Based Attacks

Only FTP and RCP are subject to header-based attacks. TFTP has a simple header structure, and if the header is invalid, the protocol will produce an error. FTP and RCP also have simple header formats, and there have not been many attacks against the header structure. There have been some attacks where the file name or directory name is too long for the FTP server to handle, which has either forced the server to hang or has forced a buffer overflow. For the most part, these vulnerabilities have been patched.

11.2.4.2 Protocol-Based Attacks

TFTP has a simple protocol and is not subject to protocol-based attacks. If there is something wrong, the TFTP client or server will time out and abort the transfer. RCP also does not have much of a protocol and just uses the TCP stream socket to transfer a file once the authentication has been established. The FTP protocol has the most vulnerabilities. This is due mostly to the complexity of the multiple data connections and how they relate to the command connection. The protocol-based attacks against FTP have primarily focused on using the data connection to bypass perimeter defenses like firewalls. Most firewalls are capable of handling the dynamic nature of the data connection by correlating the data connection requests to a current command connection.

One interesting protocol attack is to redirect the FTP server to another computer using the port command. This can be done using TELNET to connect to an FTP server, or with exploit code. The biggest trick is to get something on the FTP server you can then send to the target. Following is an example using an anonymous

FTP server that has a file (m1) that contains the email commands to interact with an SMTP server. The port command is used to direct the FTP server to connect with the computer at 192.168.1.40 on port 25, which is the email port. The FTP commands direct the FTP server to connect to the email server and send the file m1. This attack is limited in what it can do, but you could write a script that forces an FTP server to make a large number of connections to any IP address and try a denial of service, or just try to get the FTP server blacklisted. This could also be used to probe systems inside a firewall. When you try to open a connection to a port with no service running, you get an error. This method of probing is very time-consuming and not very efficient.

```
$ telnet klingon.dougj.net 21
220 klingon.doug.net FTP server ready.
user anonymous
331 Guest login ok, type your name as password.
pass doug
230 Guest login ok, access restrictions apply.
port 192,168,1,40,0,25
200 PORT command successful.
retr m1
150 Opening ASCII mode data connection for 'm1' (84 bytes).
226 Transfer complete.
quit
```



```
HELO cia.gov
MAIL FROM: badperson@cia.gov
RCPT TO: user
DATA
(any mail message)
```

11.2.4.3 Authentication-Based Attacks

The largest vulnerability associated with file transfer applications is authentication. As we saw earlier in this section, FTP uses the server authentication and prompts the user for a username and password. This provides an attacker a way to guess passwords. However, as discussed in the section on remote access, it is not a viable method to guess a large number of usernames and passwords. Anonymous

FTP can provide additional authentication problems since any user can connect to the FTP server. If there are directories that are writable, an attacker could place files on the anonymous FTP and share them with other users. As mentioned before, there are programs that scan the Internet and look for FTP servers that support anonymous access and then check for writable directories. In Figure 11.10 we can see that all the attacker needs to do is send "USER anonymous" and "PASS john@cia.gov" and see if the code 230 is returned. This will find an anonymous FTP server, and then all the attacker needs to do is send a file to the FTP server and see if the server returns an error. Since many system administrators do not closely monitor the anonymous FTP log files, these sites can often go undetected.

Another authentication problem with FTP is that the newer clients designed to run on personal computers have the option to store the username and password so that the user does not have to remember them to connect to the FTP server. This can lead to problems if a personal computer is a mobile device, is compromised, or is in an unsecured location. For example, if someone can access the computer with your FTP passwords, he or she could gain access to the FTP site. Depending on how the passwords are stored, he or she may be able to read the passwords and then use them to gain access to the server.

There are also FTP servers that can be configured to listen on any port and can be run by users. A normal FTP server needs to be executed by the privileged user since it uses a lower-numbered port and the operating system protects those ports. These user-configured FTP servers are often used to share illegal materials. There are even servers that will track the amount of data that is uploaded to the FTP sever and then allow you to download data based on the amount uploaded. This way the operator of the FTP server is asking people to contribute to the site before getting any benefit from it.

We also saw that TFTP and RCP are nonauthenticated services. RCP uses trust based on username and IP addresses. Both of these protocols should only be used when absolutely necessary, and there is a secure replacement for the RCP protocol that will be discussed in the next section.

11.2.4.4 Traffic-Based Attacks

Since all of the file transfer protocols we discussed are clear text protocols, they are subject to sniffing attacks. An attacker that can capture the traffic would also capture the username and password inside the session. An attacker can also capture all of the keystrokes and responses between the client and server. This could compromise other applications and could allow the attacker to obtain additional

usernames and passwords that the user typed during the session. The best way to mitigate sniffing vulnerabilities is through encryption. We will review several encryption methods in the next section.

FTP can be susceptible to traffic flooding attacks. This is sometimes seen in anonymous FTP sites where attackers have placed a large amount of information that is made available to the public. If a large number of people try to download very large files at the same time, the FTP server can become overwhelmed. Large file transfers can add a lot of traffic to the network and can cause reduced network bandwidth, especially through a lower-bandwidth ISP connection.

Definitions

Anonymous FTP.
An FTP server that accepts the username "anonymous" and requires no password.

User-configured FTP server.
An FTP server installed by a user that typically uses a different port number and is often used to share copyrighted material.

11.3 Peer-to-Peer Networks

The basic idea behind peer-to-peer networks is to allow users to search for files and to share files with other users without authentication. Unlike the World Wide Web, where data is stored on web sites and users can download information from the web site, peer-to-peer networks allow users to connect to each other and transfer data directly from user to user. The users of peer-to-peer networks typically do not know each other, nor do they have any relationship outside the peer-to-peer network. These networks are designed to facilitate the searching and transfer of files. Many of the peer-to-peer protocols are designed to evade detection by network filters. There are two basic types of peer-to-peer networks—centralized and decentralized (often called ad hoc)—as shown in Figure 11.12 [39–49].

The centralized peer-to-peer network uses a central server that contains an index of files that are offered by the users of the network. The files are stored on the users' computers and not on the central server. Users send a list of files they have to share to the central index server. Users query the central index server to find a file that matches their search criteria. Files are transferred from one user

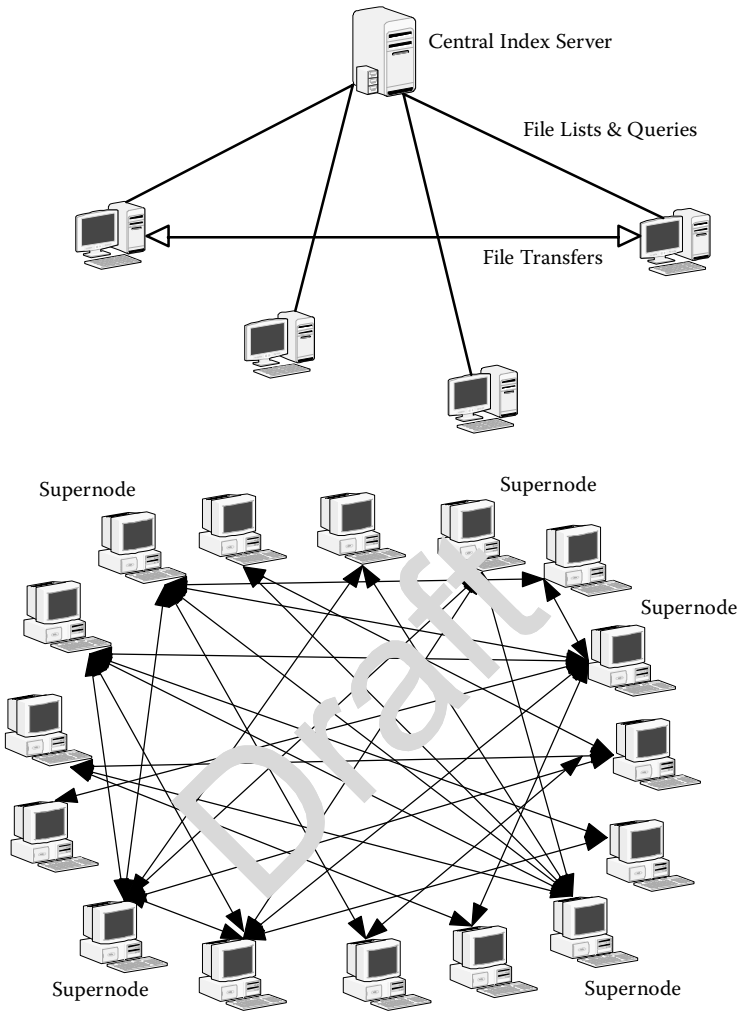


Figure 11.12: Peer-to-peer networks.

to another user. In addition, users can connect directly to each other and search for files without the central index server. The central index server simply makes searching easer. The central index server model can have more than one index server interconnected in its own peer-to-peer network. This is done to speed up searches and distribute the workload between several servers. This also provides redundancy, so that if one server node quits, the other nodes can still function and the network is still usable.

With an ad hoc peer-to-peer network, every computer that is part of the network has its own list of files that are shared, and each computer is connected to a small number of other computers (neighbors). Each neighbor is connected to a small number of computers, and so on. When a user wishes to search for a file, a request is sent to each neighbor, and each neighbor sends the request to the next neighbor, and so on. When a computer gets the search request, it searches its shared files, and if there is a match, it will send a message back to the requester telling him or her that it has the file(s) and some basic information about the files.

The ad hoc network can have different types of nodes, often called leaf nodes and super nodes. A leaf node is often a user's computer, and the super nodes are designed to have a large number of leaf nodes connected to them. This way the distance between any two users (number of peer-to-peer nodes between two computers) is reduced, thus making the network faster.

In this section we will look at three different protocols used to support peer-to-peer networks. We will also look at the vulnerabilities in peer-to-peer networks and methods used to block peer-to-peer networks.

11.3.1 Centralized Peer to Peer

There are two popular centralized peer-to-peer protocols. The first is called Napster and was one of the first centralized peer-to-peer networks [50–58]. The second centralized peer-to-peer network is called KaZaA [59–70]. KaZaA runs a protocol called Fasttrack. Both Napster and KaZaA were designed to share music between users and were established as free music sharing systems. Both Napster and KaZaA have converted to a pay-per-song model, and the Napster protocol is not widely used. There are several applications that still use the Fasttrack protocol to share files without paying for them. Both protocols are very similar to each other. The primary difference is that in the Napster protocol the central index server keeps a record of every file transfer, and in the Fasttrack protocol the central index server does not. The Napster protocol is shown in Figure 11.13.

Napster uses TCP between the client and the central index server. In the figure we see the packet format is simple, consisting of a 2-byte-long field, which is the length of the packet. The type field is 2 bytes long and contains a number that indicates the packet type. The content of the data depends on the type of packet. The client logs in to the server and notifies the server which files it has to share. The client can then issue search requests and the server will return the results. The results contain information about the files that matched the search

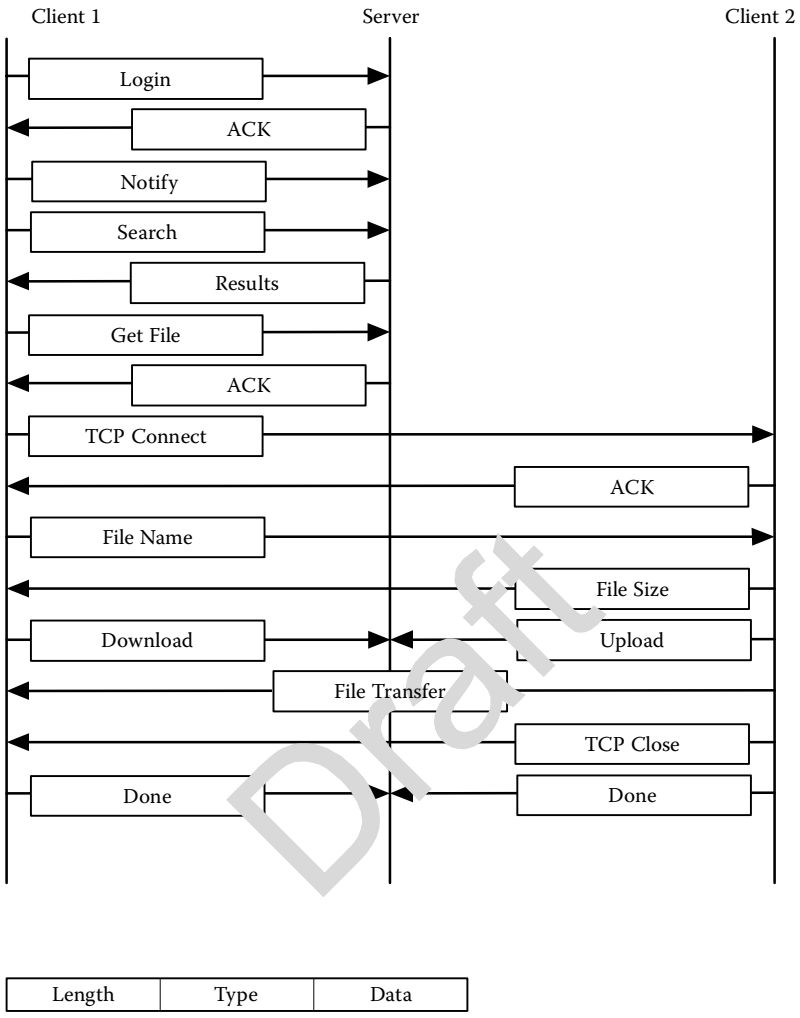


Figure 11.13: Napster protocol.

along with the location (IP address) where the file can be found. When a client wishes to retrieve a file it issues a get packet to the index server and the index server responds with an ACK packet. The client then connects to the peer client that has the file. The two clients exchange packets to determine which file should be transferred. The file transfer then takes place using the TCP connection that was established between the two peer clients. Both clients notify the central index server that a file transfer has been started. The peer clients notify the central index server when the file transfer is complete. Napster can also handle the case where

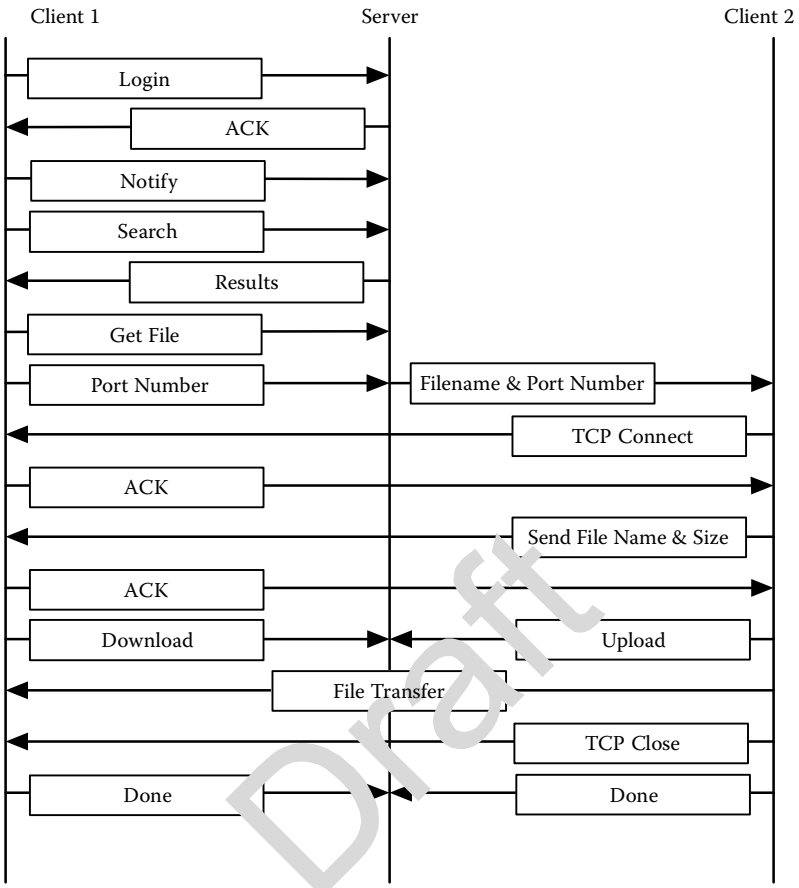


Figure 11.14: Napster protocol behind a firewall.

one client is behind a firewall that blocks incoming connections. Figure 11.14 shows the Napster protocol with a firewall.

Figure 11.14 shows the protocol assuming client 2 is behind a firewall, which means client 1 cannot connect to it. Client 2 tells the server when it connects to the server if it is behind a firewall. Client 1 tells the server which port number it will use to wait for a connection. The server tells client 2 which port number to use to connect to client 1, and when client 2 connects, it tells client 1 which file it is sending. The remainder of the protocol works the same way as shown in Figure 11.13.

Napster was the first music sharing peer-to-peer network to lose a lawsuit brought by the record companies. This was due in part to the fact that the central

index server keeps track of what files are transferred. Napster could not claim it just provided a protocol. Several other companies that produced peer-to-peer software have also been forced to change how they operate due to the recording industry suits. The Fasttrack protocol is still widely used today to share copy-righted material. The Fasttrack protocol is best known by the application KaZaA, which has changed its business model to charge for content.

11.3.2 KaZaA

KaZaA is a centralized index server peer-to-peer network based on the Fasttrack protocol. KaZaA has a shared folder that is uses to store files that are downloaded from other users. By default, this shared folder is located in the KaZaA program directory. KaZaA also provides the ability for the user to set up additional sharing folders that are used to share files with other KaZaA users. When a user starts KaZaA, he or she is connected to a super node and KaZaA offers or advertises the files it has to share. Figure 11.15 shows the Fasttrack protocol.

To advertise a file the user must put the files into either the download folder or one of the additional sharing folders that KaZaA uses. Any files within these shared folders are advertised and can be downloaded by other users of the KaZaA network. The advertisement consists of a set of identifiers that are used to tie the file back to the users. These identifiers include the IP address of the client offering the files, the name of the file, the file size, and the content hash. In addition, there are file descriptors that provide information like the artist name, album name, and user text. This information is used in the search process. The user text field is used to provide a description of the files and is part of the KaZaA system. The content hash is a mathematical function that is used to identify files that are the same. This allows the user to search for the file if the original file download fails.

To find a file, the user submits a query to the super node. The super node looks in its database for the file that matches the search parameters. If one or more of the users connected to the super node has the file that matches the request, then the super node returns the IP addresses and the file descriptions of all matches. Super nodes can send queries between each other. Users can also connect directly with each other, so if you find a file on a user's machine, you can then query it to see what other files are offered for sharing. The file transfer takes place between the two clients using the HTTP protocol.

Unlike Napster, in the Fasttrack protocol the index server does not record which files are transferred between clients.

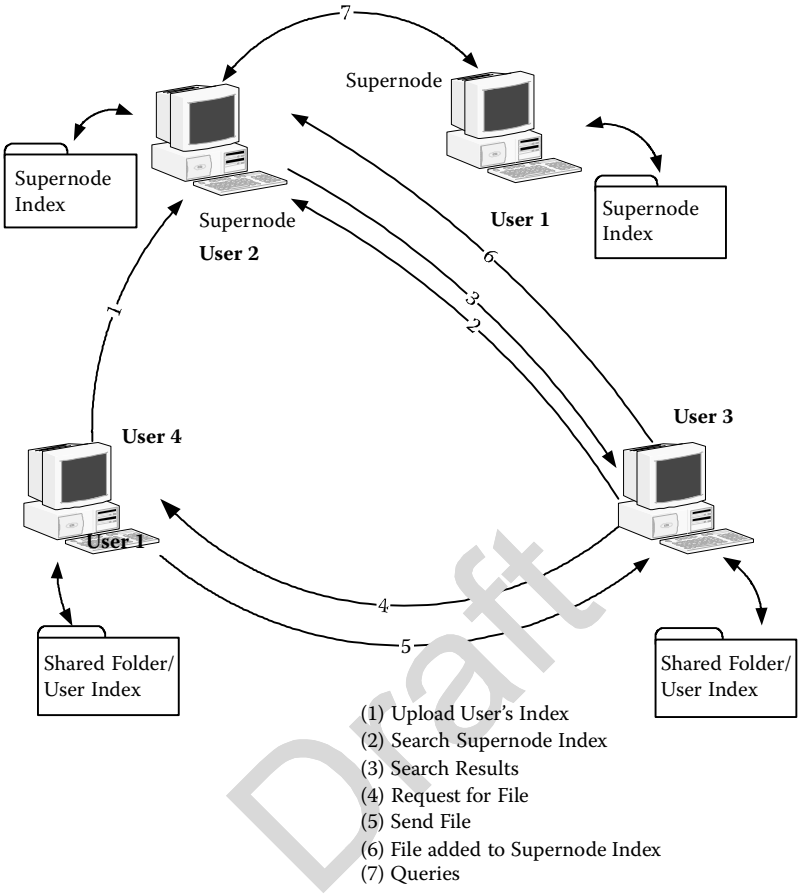


Figure 11.15: Fasttrack protocol.

11.3.3 Decentralized Peer to Peer

The first widely used decentralized peer-to-peer protocol is called Gnutella and is still in use today [71–78]. There are several applications that have been designed to use the Gnutella protocol that run on most major operating systems. Unlike centralized peer-to-peer protocols, Gnutella applications need to discover their neighbors and all searches flow through the applications that are part of the Gnutella network. Another decentralized peer-to-peer network is called BitTorrent [71–83]. BitTorrent is designed to allow the shared file to be split among multiple clients. This is done to speed up the download of large files like movies and CD-ROM images. We will not examine the BitTorrent protocol as part of this

book. The vulnerabilities that exist with BitTorrent are the same as those of other peer-to-peer protocols. The mitigation methods are also the same.

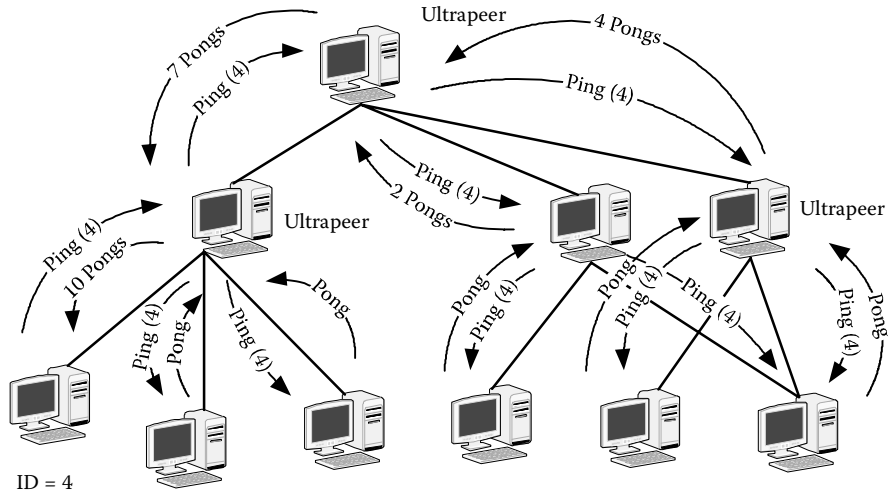
11.3.3.1 Limewire, Bearshare, and Gnutella

Limewire, Bearshare, and Gnutella are all ad hoc peer-to-peer networks. Gnutella is the protocol used by applications like Limewire. These applications have a shared folder that is used to store files that are downloaded from other users. When a user starts a Gnutella-based program, she is connected to a Gnutella network.

To advertise a file the user must put the file into either the download folder or one of the additional sharing folders that she sets up. Any files within these shared folders are advertised and can be downloaded by other users of the Gnutella network. The advertisement consists of a set of identifiers that are used to tie the file back to the user. These identifiers include the IP address of the client offering the files, the name of the file, the file size, and the content hash. In addition, there are file descriptors that provide information like the artist name, album name, and user text. This information is used in the search process. The user text field is used to provide a description of the file and is part of the Gnutella system. This field is not part of the original data stored on a CD and is added by users who put files into the Gnutella shared folders. The content hash is a mathematical function that is used to identify files that are the same. This allows the user to search for the file if the original file download fails. Figure 11.16 shows a typical Gnutella network.

Figure 11.16 shows several nodes connected to neighbor nodes using the Gnutella protocol. An issue with ad hoc peer-to-peer networks is how to create and maintain the network. Gnutella uses a protocol based on sending out "ping" packets and then waiting for responses (called a pong). The protocol is shown in Figure 11.16.

The packet format is simple (as shown in Figure 11.6). The payload indicates what type of packet it is, the time-to-live (TTL) field indicates the lifetime of the packet, and the hop count indicates how far the packet has traveled. Just like in an IP protocol, the TTL is decremented by each client that forwards the packet. The hop count starts out at zero and is incremented by each client that forwards the packet. Figure 11.16 shows the client sending out a ping packet with an ID of 4 to its neighbor nodes. When the ping packet is received by a neighbor, the neighbor forwards the ping packet on to its neighbors and also responds back to the originator with a pong packet. The ping packet will continue to be forwarded from neighbor to neighbor until the TTL equals zero. Each neighbor that gets a unique ping packet (the ID field is used to tell if the neighbor has seen the same



ID	Payload	TTL	Hop	Length	Data

Gnutella Packet

Payload	
00	Ping
01	Pong
80	Query
81	Query Hit

Port	IP	Number of Files Shared	Number of Bytes Shared

Pong Packet

Min Speed	String

Query Packet

Hits	Port	IP	Speed	Results	IP

Query-Hit Packet

Figure 11.16: Typical Gnutella network.

ping packet more than once) sends a pong back to the originator. The pong packet is returned following the same path as the ping packet. This creates a large amount of traffic between neighbors. The pong packet contains the port number of the responding application, the IP address of the application, the total number of files shared, and the total number of kilobytes of shared data. As we see in the figure, the single ping packet generated multiple ping packets and ten pong packets were sent back in response to the single ping.

Figure 11.16 also shows the query and query-hit packets. A client wishing to find a file sends out a query packet to its neighbors. The neighbors pass the packet on to their neighbors, and this continues until the maximum number of hops is reached. Each application that gets a query searches through its shared files to determine if there is a match. If there is a match, the application responds using the query-hit packet. The response is routed back through the Gnutella network until it reaches the original sender. The client that sent the query can then chose which result to use to get the file. The file transfer takes place between the client with the file and the client wanting the file. The clients use the HTTP protocol to transfer the file.

11.3.4 Vulnerabilities, Attacks, and Countermeasures

There are numerous vulnerabilities associated with peer-to-peer networks. In addition to those vulnerabilities associated with the taxonomy there are legal issues with sharing copyrighted materials. Several copyright holders have started programs to identify and prosecute people who share copyrighted material using peer-to-peer networks. The network-based peer-to-peer vulnerabilities and attacks are described in the next sections.

11.3.4.1 Header- and Protocol-Based Attacks

The headers in the peer-to-peer networks are simple, and any header-based vulnerabilities would be limited to disabling the communication of the client. The centralized protocols could be subject to header-based attacks against the central index server that could disable the index server. Likewise, any protocol-based attacks would be limited to clients or the index server. An attacker has very little to gain from a header- or protocol-based attack since these applications run as a user and do not provide access to the system.

11.3.4.2 Authentication-Based Attacks

Peer-to-peer networks are designed to be anonymous, and therefore the concept of authentication does not apply. This leads to several security problems, primarily focused around the validity of the content. In all of the peer-to-peer protocols we have examined, the files to be shared are reported by the clients with the files. Since anyone can join a peer-to-peer network, there is no checking as to the validity or intentions of the person offering the files for sharing. This leads to attackers sharing files that contain viruses or other malicious code. These files are often named to match a desired file. Corrupted files are also put out in the network

so when a client downloads the file, it is not valid. This sometimes happens with copyrighted material in an effort to slow down file sharing.

A common misconception about peer-to-peer networks is that they are only used to share music. In fact, they share all files within the directories the user specifies to be shared. A user could share the contents of his entire hard drive. This can lead to accidental sharing of files that the user did not intend to be shared.

Another problem with peer-to-peer networks is the perceived feeling of privacy when in fact the users of peer-to-peer networks can be traced and some of their activity can be monitored. Even though the peer-to-peer network does not require authentication, the IP address of the user with the files to be shared needs to be known in order to transfer files from the user. Someone could do a search and, based on the results, transfer a file and determine the IP address of the computer that has the file. Also, when users issue a search, the search query goes through either a central index server or the neighbors in a decentralized network. It would be possible to capture the search strings, but not possible (unless you are part of the file transfer) to determine if a file was transferred as the result of a search. However, the search data could be used tell what someone is looking for, which could be embarrassing to the user.

11.3.4.3 Traffic-Based Attacks

Peer-to-peer networks can generate large amounts of traffic either because they encourage large file transfers or because the protocol itself generates a large number of packets. The decentralized protocols like Gnutella route traffic through the clients, which means that even if a client is not transferring files there can still be a large amount of traffic. Any devices that have large files offered for sharing can also create traffic problems. Super nodes and index servers can also generate a larger amount of traffic without transferring files.

Sniffing of the traffic is possible in most peer-to-peer networks. This is not much of a problem since the applications are not authenticated. Some peer-to-peer networks use encryption, but this is used to help prevent any countermeasures from examining the data.

11.3.4.4 Peer-to-Peer Countermeasures

Many of the peer-to-peer protocols have been designed to avoid detection and evade normal countermeasures. Network-based peer-to-peer countermeasures

need to examine the TCP payload to determine if the TCP connection is being used to transport peer-to-peer traffic. There are two types of devices on the market designed to detect and mitigate the effects of peer-to-peer networks: an inline device and a passive device. Both types need to examine the payload and take action based on the packet exchange. The action can be to block, log, or in some cases reduce the bandwidth of the offending traffic.

An inline device will operate at the physical network layer, and therefore does not need to be configured like a router. There are routers that can also provide peer-to-peer detection and mitigation. A passive device sniffs the traffic and uses the TCP reset packet to terminate the connection, as described in Chapter 7. The advantage of the passive device is that there is no latency. Another aspect of the countermeasure is to limit the bandwidth of the peer-to-peer protocols. This is best done with the inline device and involves manipulating the TCP window size. Bandwidth reduction only mitigates the traffic-based vulnerabilities of peer-to-peer protocols, since the protocol is still allowed to function.

Definitions

Central index server (super node).
The server that supports a centralized P2P network.

Centralized P2P network.
A P2P network where each user connects with a central server. The central server keeps an index of the files each user is sharing. Users send search requests to the server and tell the requester which user has the file.

Decentralized P2P network.
A P2P network where each user connects to several neighbors to create a large network of interconnected users. Searches are propagated from computer to computer through the network. The identity of the computer with the file is returned to the requester. The requester can then get the file directly from the source.

Peer-to-peer (P2P) network.
An application that allows a group of users to connect with each other for the purpose of searching for files and exchanging files.

Ultrapeer.
A computer in the Gnutella P2P network that supports a large number of clients. This speeds up the searches.

11.4 General Countermeasures

As we saw in the discussion concerning vulnerabilities with both remote access protocols and file transfer protocols, the two biggest issues were authentication and clear text data transfer. In this section we will examine several protocols that can help mitigate these issues.

11.4.1 Encrypted Remote Access

When we look at the two most common issues (authentication and clear text), an obvious solution comes to mind, which is to encrypt the traffic and use encryption keys to help verify authenticity of the hosts. There are several methods to provide encrypted traffic, including encryption at the lower layers. For example, wireless networks support encrypted traffic. The most complex issue with network encryption is key distribution and key authentication. Appendix A provides an overview of data encryption and key distribution.

There are several methods commonly used to provide data encryption of remote access protocols. These methods can be grouped into two categories: application-based encryption and tunnel-based encryption. Application-based encryption is where the application protocol includes the encryption functionality. For example, this would include secure TELNET. Tunnel-based encryption uses a software wedge (and sometimes hardware to support the tunnel) to create an encrypted tunnel where the application can function without alteration. Transport Layer Security/Secure Sockets Layer (TLS/SSL) is an example of tunnel-based encryption at the transport layer, and virtual private networks (VPNs) are an example of tunnel-based encryption at the IP layer. Figure 11.17 shows the difference between the two methods.

Both methods provide encrypted traffic across the network. The tunnel-based method can handle multiple application types. It is also possible to combine the two methods. For example, an application with built-in encryption can connect to a tunnel-based layer as long as it understands the protocol.

All of the protocols have several common features that are outlined in Figure 11.18. Protocols start out with a protocol identification and key negotiation phase. This is followed by an optional user authentication phase where the application may require the user to authenticate his or her identity. This is followed by the data transfer phase and then session termination phase. There are some protocols that might change the keys during the data transfer phase. Typically the key negotiation phase consists of two parts, where the two sides

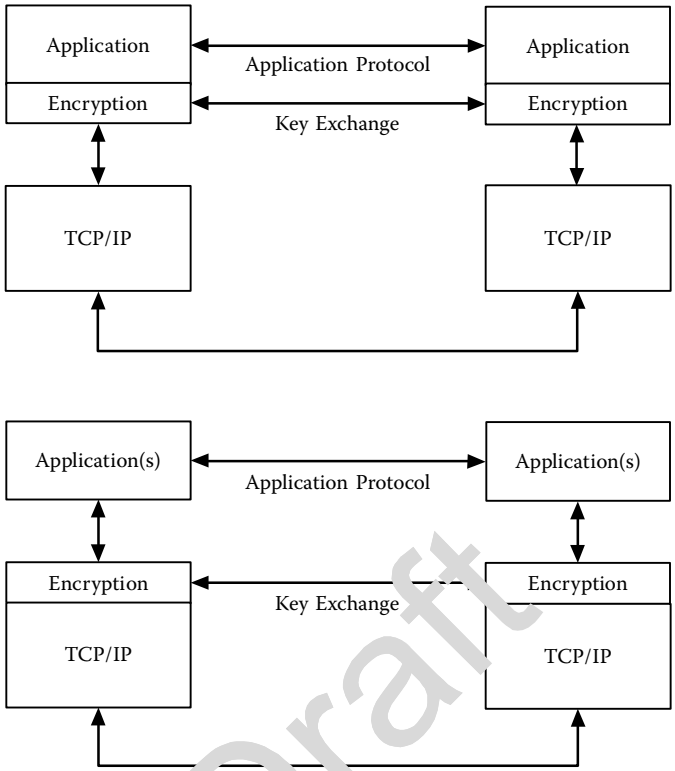


Figure 11.17: Application-based versus tunnel-based encryption.

use a shared secret to negotiate a one-time session key that is used for the data transfer.

In the next couple of sections we will examine a few protocols that are in common use today. These protocols are either replacements or enhancements for the remote access protocols described in this chapter.

11.4.2 SSH

Secure Shell (SSH) is an open source protocol that supports machine authentication and encrypted traffic [84–90]. With clients and servers available for most operating systems, SSH has become a de facto standard for secure remote access. SSH is modeled after the rlogin remote access protocol in that it does very little translation of data. Any data translation is left up to the client and server. SSH can also support tunneling of other protocols, including X-Windows. In this section we will examine how SSH functions and look at a few vulnerabilities that exist.

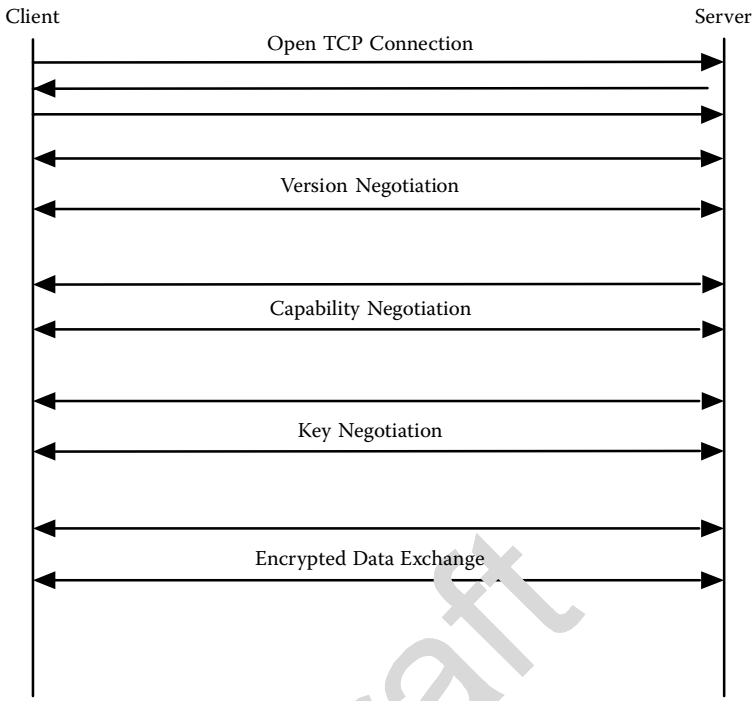


Figure 11.18: Encrypted remote access protocols.

SSH is designed to use both public and symmetric key encryption (see Appendix A for a description of encryption methods). The public key encryption is used to authenticate the SSH server, and the symmetric key is used to encrypt the data transfer. Figure 11.19 shows the protocol exchange for an SSH session. Note that the exact packet format and data payload value are not shown or discussed. This exploration is left up to the reader as a set of lab experiments included at the end of this chapter.

In Figure 11.19 the server starts by announcing its version, and the client responds. Then the server and client exchange capabilities and preferences. The client and server negotiate a session key based on the key exchange preferences. The session key is used to encrypt the traffic exchange between the client and server. Once the client and server have negotiated a session key, the user can be authenticated. The application authentication is handled outside the SSH protocol and is treated as data by SSH.

SSH supports server machine authentication by using the server's public key as an identifier. When a client makes contact with a server, it checks to see if

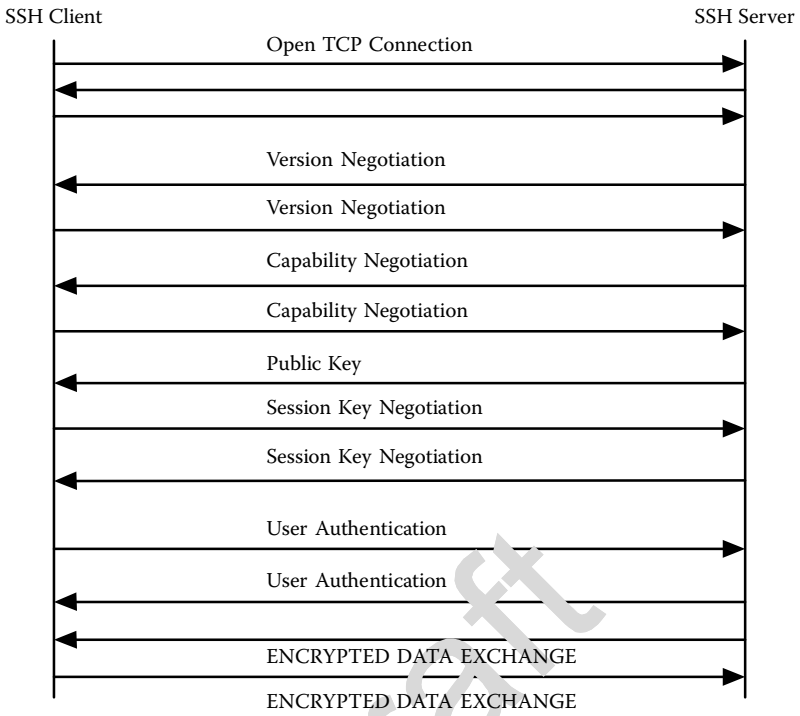


Figure 11.19: SSH protocol exchange.

the server's public key is known to the client. If it is the first time the client is connecting to the server, the user will be prompted to verify the server public key. Once the server public key has been verified, the next time the client connects to that server the advertised public key will be compared to the public key stored on the client, and if they match, the client finishes the key exchange. If the transmitted server public key does not match the server public key stored on the client, then the client does not connect. This can lead to a problem if the server public key changes. The server public key can change if the server is upgraded or the server suffers a system crash and must be rebuilt. The user then needs to clear the old server public key so the client software thinks that it is talking to the server for the first time and stores the new key for the server.

The biggest weakness in SSH is a man-in-the-middle attack. This type of attack is complex to carry out since it requires convincing the client that the attacker is the server. This can be done with DNS or ARP poisoning. However, if the client has connected with the server before the attack, the attacker's public key

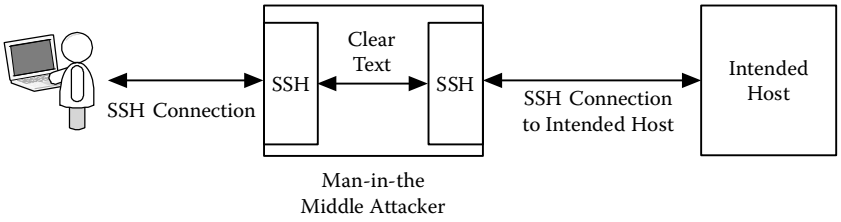


Figure 11.20: SSH man-in-the-middle attack.

would not match the stored public key of the server and the connection would be rejected. If the attacker manages to fool the client user, then the attacker just needs to open an SSH connection with the intended server and pass all traffic from the user to the intended server. As the traffic passes through the attacker, it would be in clear text. Figure 11.20 illustrates a successful man-in-the-middle attack.

11.4.3 Remote Desktop

Remote Desktop (RDP) is a protocol and application used by Microsoft to enable clients to connect to Microsoft Windows–based servers [91–93]. This can be thought of as X-Windows for Microsoft. For the purpose of this book, we are not going to examine the protocol in detail, nor will we look at the features supported by the applications. Unlike X-Windows, Remote Desktop supports encryption and user authentication. The original versions of Remote Desktop use the encryption that is part of the RDP protocol, which is modeled after the T.120 set of protocols. The RDP protocol uses a symmetric key encryption algorithm (RC4) for the one-time session key. The one-time session key is exchanged after the client and server exchange public keys. This exchange is very similar to the method used by SSH. The RDP protocol supports three RC4 key sizes that correspond to three different security levels:

High-level security: Uses a 128-bit key to encrypt the data in both directions.

Medium-level security: Uses a 56-bit key (or 40-bit, depending on the Operating System (O.S.) version of the client) to encrypt the data in both directions.

Low-level security: Uses a 56-bit (or 40-bit) key to encrypt the data from the client to the server.

Remote Desktop is vulnerable to session key decoding if the security level is low or medium. Even with high-level security it might be possible to guess the key. In order to guess the session key, the attacker needs to be able to capture the network traffic in an open wireless environment. The low-level security is really only used to protect the password, since all data from the server to the client is in clear text.

Newer versions of Remote Desktop can use TLS to provide data encryption. This provides a more secure connection between the client and the server. Remote Desktop is vulnerable to a man-in-the-middle attack, just like SSH, and as with the man-in-the-middle attacks against SSH, it takes several things to happen at once for the man-in-the-middle attacks on RDP to be successful.

11.4.4 Secure File Transfer (SFTP, FTPS, HTTPS)

There are several protocols that have been designed to replace FTP for encrypted file transfer. These protocols are designed to protect the authentication exchange and the data transfer from eavesdropping. Most of these protocols build on either SSH or SSL encryption protocols. In this section we will briefly examine the most common secure file protocols by looking at what underlying secure data transfer protocol they use and how they differ from each other.

SSH File Transfer Protocol (SFTP) is an FTP-like client and server in that it supports the same types of commands and functions that are supported by FTP. SFTP uses the SSH protocol to encrypt both the command channel and the data channel. SFTP is not simply FTP operating over an SSH tunnel, but a new protocol that is based on the SSH protocol. SFTP requires both a client and server that understand the protocol. SFTP has clients and servers available for most operating systems. The SFTP protocol is vulnerable to a man-in-the-middle attack, but this attack would be even more difficult to carry out than an SSH man-in-the-middle attack since the protocol is much more complex. A man-in-the-middle attack on SFTP would work best to obtain usernames and passwords.

File Transfer Protocol/SSC (FTPS) is a secure version of the FTP protocol that uses the SSL/TLS secure transport layer protocols. FTPS is an enhancement to the standard FTP protocol and, using the secure features, is something that can be negotiated during the initial connection phase using the AUTH TLS command. The FTPS server waits on port 21 like the standard FTP server and can operate like a standard FTP server. SSL and TLS were described earlier in the book, and therefore we will not reexamine them now.

HTTP over SSL (HTTPS) is another method used for secure file transfer. This happens through a secure web site when you click on a link that contains a file. The file is transferred over the secure connection. The reader is directed to Chapter 10 on the web to understand how HTTPS works and its vulnerabilities.

Homework Problems and Lab Experiments

Homework Problems

1. Research the RFCs associated with TELNET and comment on the changes in the protocol over the years.
2. Research the RFCs associated with the security extensions of TELNET.
3. Comment on the security of running a TELNET server on a host.
4. Comment on the security of running or using the TELNET client.
5. Show the packet exchange for a user to log in to a TELNET server using the username "bob" and the password "alice." Compute the overhead involved during the exchange (total number of bytes in the packets versus the actual payload).
6. Show the packet exchange for a user to log in to an rlogin server using the username "bob" and the password "alice." Compute the overhead involved during the exchange (total number of bytes in the packets versus the actual payload).
7. Research the RFCs associated with the FTP protocol.
8. Research the Secure Copy Protocols (SCPs).
9. Research different peer-to-peer applications. Develop a table showing the different protocols used versus the actual application names. Provide comments on each protocol type as to how hard it would be to detect the protocol.
10. Research the legal battles between the copyright holders and the various companies and groups that create peer-to-peer software.
11. Research several P2P filtering products and determine how they operate. Compare P2P filtering products with bandwidth shaping products.

12. Research the SSH protocol and the attacks against it. Comment on the likelihood of the various attacks being successful.

13. Research the RFCs associated with the two versions of secure FTP (FTPS, SFTP). Comment on the differences between the two protocols. Why do you think there are two protocols?

Lab Experiments

1. Use TELNET to connect to various applications (web, FTP, email, rlogin, etc.). Comment on how attackers could use what they find by using TELNET to connect to each application.

2. Use tcpdump or wireshark to capture a TELNET session between two machines in the lab. Issue several commands to generate traffic.

   a. Look at the network capture and find the username and password.

   b. Look for the command and the results of the commands in the network traffic.

   c. Comment on what you see.

3. Use tcpdump or wireshark to capture an rlogin session between two machines in the lab. Issue several commands to generate traffic.

   a. Look at the network capture and find the username and password.

   b. Look for the command and the results of the commands in the network traffic.

   c. Comment on what you see.

4. Use tcpdump or wireshark to capture an FTP session between two machines in the lab. Issue several commands to generate traffic.

   a. Look at the network capture and find the username and password.

   b. Look for the command and the results of the commands in the network traffic.

   c. Comment on what you see.

5. Use tcpdump or wireshark to capture an X-Windows session between two machines in the lab. Issue several commands to generate traffic.

a. Look for the command and the results of the commands in the network traffic.

b. Comment on what you see.

6. Use tcpdump or wireshark to capture a SSH session between two machines in the lab. Issue several commands to generate traffic.

   a. Look at the network capture and find the clear text parts of the session.

   b. Comment on the protocol exchange and the overhead involved in both initial connection and the data transfer.

7. Use tcpdump or wireshark to capture a Remote Desktop session between two machines in the lab. Issue several commands to generate traffic.

   a. Look at the network capture and find the clear text parts of the session.

   b. Comment on the protocol exchange and the overhead involved in both initial connection and the data transfer.

8. Use tcpdump or wireshark to capture peer-to-peer traffic in the test lab.

   a. Look at the network capture and find the clear text parts of the session.

   b. Comment on the protocol exchange and the overhead involved in both initial connection and the data transfer.

Programming Problems

1. Use the code you downloaded for Chapter 5 and modified in Chapters 6, 9, and 10. Add code to perform the following:

   a. Add a flag to the program (-p) that will disable printing of all header information and will only look for usernames and passwords in the FTP and TELNET protocols.

   b. Modify the program to print usernames and passwords it finds along with the IP address of the machines.

   c. Use the program to search for usernames and passwords on the network.

   d. Comment on how this program could be used and how it could be placed on a network.

   e. Comment on any possible countermeasures for this program.

2. Use the code you downloaded for Chapter 5 and modified in Chapters 6, 9, and 10, and in programming problem 1. Add code to perform the following:

   a. Decode and print TELNET and FTP payload. Print the payload in ASCII.

   b. Add to the set of counters a counter for the number of TELNET packets and the number of FTP packets. Add code to print the values of these counters to the subroutine program_ending().

   c. Add to the set of counters a counter for the number of SSH packets and the number of X-Windows packets. Add code to print the values of these counters to the subroutine program_ending().

3. Use the code you downloaded for Chapter 5 and modified in Chapters 6, 9, and 10, and in programming problems 1 and 2. Add code to perform the following:

   a. Add a flag to the program (-s) that will disable printing of all header information and will only look for ASCII characters in the payload of all packets and will print the characters.

   b. Add to the set of counters a counter for the number of ASCII bytes found in the traffic. Add code to print the values of this counter to the subroutine program_ending(). This counter should be updated and printed independent of the value of the flag (-s).

   c. Add to the set of counters a counter for the number of total bytes found in the traffic. Add code to print the values of this counter to the subroutine program_ending(). This counter should be updated and printed independent of the value of the flag (-s).

References

[1] Page, J. 1986. Kermit: A file-transfer protocol. *Accounting Review* 61: 368–69.

[2] Walters, W. 1987. Implementing a campus-wide computer-based curriculum. In *Proceedings of the 15th Annual ACM SIGUCCS Conference on User Services*, Kansas City, MO: 465–68.

[3] Banks, M. A. 2000. *The modem reference*. Medford, NJ: Cyberage Books.

[4] Khare, R. 1998. Telnet: The mother of all (application) protocols. *IEEE Internet Computing* 2:88–91.

[5] Borman, D. 1994. *Telnet environment option interoperability issues*. RFC 1571.

[6] Altman, J., and T. Ts'o. 2000. *Telnet authentication option*. RFC 2941.

[7] Murphy, Jr., T., P. Rieth, and J. Stevens. 2000. *5250 Telnet enhancements*. RFC 2877.

[8] Hedrick, C. L. 1988. *Telnet remote flow control option*. RFC 1080.

[9] Postel, J., and J. K. Reynolds. 1983. *Telnet protocol specification*. RFC 0854.

[10] Leiner, B., et al. 1985. The DARPA Internet protocol suite. *IEEE Communications Magazine* 23:29–34.

[11] Tam, C. M. 1999. Use of the Internet to enhance construction communication: Total information transfer system. *International Journal of Project Management* 17:107–11.

[12] Day, J. 1980. Terminal protocols. *IEEE Transactions on Communications* 28:585–93.

[13] Cohen, D., and J. B. Postel. 1979. On protocol multiplexing. In *Proceedings of the Sixth Symposium on Data Communications*, Pacific Grove, CA: 75–81.

[14] Kantor, B. 1991. *BSD rlogin*. RFC 1282.

[15] Kantor, B. 1991. *BSD rlogin*. RFC 1258.

[16] Bahneman, L. 1994. The term protocol. *Linux Journal* 1994(8es).

[17] Stevens, W. R. 1994. *TCP/IP illustrated*. Reading, MA: Addison-Wesley Professional.

[18] Rogers, L. R. 1998. *Rlogin (1): The untold story*. NASA.

[19] Uppal, S. 1989. Performance analysis of a LAN based remote terminal protocol. In *Proceedings of the 14th Conference on Local Computer Networks*, Minneapolis, MN: 85–97.

[20] Stevens, W. R. 1995. *TCP/IP illustrated*. Vol. I, 223–27. Upper Saddle River, NJ: Addision Wesley Publishing Company.

[21] Scheifler, R. W., and J. Gettys. 1986. The X window system. *ACM Transactions on Graphics (TOG)* 5:79–109.

[22] Richardson, T., et al. 1994. Teleporting in an X window system environment. *IEEE Personal Communications Magazine* 1:6–12.

[23] Quercia, V., and T. O'Reilly. 1993. *X window system user's guide*. Sebastopol, CA: O'Reilly.

[24] Nye, A. 1995. *X protocol reference manual*. Sebastopol, CA: O'Reilly.

[25] McCormack, J., and P. Asente. 1988. An overview of the X toolkit. In *Proceedings of the 1st Annual ACM SIGGRAPH Symposium on User Interface Software*, 46–55.

[26] Scheifler, R. W., et al. 1990. *X-window system: The complete reference to XLIB, X protocol, ICCCM, XLFD: X version 11, release*.

[27] Postel, J., and J. Reynolds. 1985. *File transfer protocol (FTP)*. STD 9, RFC 959.

[28] Horowitz, M., and S. Lunt. 1997. *FTP security extensions*. RFC 2228.

[29] Bellovin, S. 1994. *Firewall-friendly FTP*. RFC 1579.

[30] Bhushan, A. K. 1973. *FTP comments and response to RFC 430*. RFC 0463.

[31] Bhushan, A., et al. 1971. *The file transfer protocol*. RFC 0172.

[32] Neigus, N. 1973. *File transfer protocol*. RFC 0542.

[33] Sollins, K. 1992. *The TFTP protocol* (revision 2). RFC 1350.

[34] Emberson, A. 1997. *TFTP multicast option*. RFC 2090.

[35] Malkin, G., and A. Harkin. 1998. *TFTP option extension*. RFC 2347.

[36] Aslam, T., I. Krsul, and E. Spafford. 1996. Use of a taxonomy of security faults. Paper presented at the 19th National Information Systems Security Conference Proceedings, Baltimore.

[37] Stevens, W. R., and T. Narten. 1990. Unix network programming. *ACM SIGCOMM Computer Communication Review* 20:8–9.

[38] Stevens, W. R. 1994. *TCP/IP illustrated*. Vol. 1. *The protocols*, chap. 15. Reading, MA: Addison Wesley.

[39] Golle, P., K. Leyton-Brown, and I. Mironov. 2001. Incentives for sharing in peer-to-peer networks. *Electronic Commerce* 14:264–67.

[40] Tran, D. A., K. A. Hua, and T. T. Do. 2004. A peer-to-peer architecture for media streaming. *IEEE Journal on Selected Areas in Communications* 22:121–33.

[41] Androutsellis-Theotokis, S., and D. Spinellis. 2004. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys (CSUR)* 36:335–71.

[42] Androutsellis-Theotokis, S. 2002. A survey of peer-to-peer file sharing technologies. Athens University of Economics and Business.

[43] Ramaswamy, L., and L. Liu. 2003. Free riding: A new challenge to peer-to-peer file sharing systems. Paper presented at Proceedings of the Hawaii International Conference on Systems Science. Big Island, HI.

[44] Lui, S. M., and S. H. Kwok. 2002. Interoperability of peer-to-peer file sharing protocols. *ACM SIGecom Exchanges* 3:25–33.

[45] Gummadi, P. K., S. Saroiu, and S. D. Gribble. 2002. A measurement study of Napster and Gnutella as examples of peer-to-peer file sharing systems. *ACM SIGCOMM Computer Communication Review* 32:82.

[46] Daswani, N., H. Garcia-Molina, and B. Yang. 2003. Open problems in data-sharing peer-to-peer systems. In *Proceedings of the 9th International Conference on Database Theory*, Sienna, Italy: 1–15.

[47] Christin, N., A. S. Weigend, and J. Chuang. 2005. Content availability, pollution and poisoning in file sharing peer-to-peer networks. In *Proceedings of the 6th ACM Conference on Electronic Commerce*, San Diego, CA: 68–77.

[48] Yang, B., and H. Garcia-Molina. 2002. Improving search in peer-to-peer networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, Vienna, Austria: 5–14.

[49] Kant, K. 2003. An analytic model for peer to peer file sharing networks. In *IEEE International Conference on Communications (ICC'03)*, Anchorage, AK: 3.

[50] Saroiu, S., K. P. Gummadi, and S. D. Gribble. 2003. Measuring and analyzing the characteristics of Napster and Gnutella hosts. *Multimedia Systems* 9:170–84.

[51] Scarlata, V., B. N. Levine, and C. Shields. 2001. Responder anonymity and anonymous peer-to-peer file sharing. In *Ninth International Conference on Network Protocols*, Riverside, CA: 272–80.

[52] Aberer, K., and M. Hauswirth. 2002. An overview on peer-to-peer information systems. Paper presented at Workshop on Distributed Data and Structures (WDAS-2002). Paris, France.

[53] Moro, G., A. M. Ouksel, and C. Sartori. 2002. Agents and peer-to-peer computing: A promising combination of paradigms. In *Proceedings of the 1st International Workshop of Agents and Peer-to-Peer Computing (AP2PC2002)*, Bologna, Italy. 1–14.

[54] Howe, A. J. 2000. Napster and Gnutella: A comparison of two popular peer-to-peer protocols. Department of Computer Science, University of Victoria, British Columbia, Canada.

[55] Braione, P. 2002. A semantical and implementative comparison of file sharing peer-to-peer applications. In *Proceedings of the Second International Conference on Peer-to-Peer Computing (P2P 2002)*, Linköping, Sweden: 165–66.

[56] Fellows, G. 2004. Peer-to-peer networking issues—An overview. *Digital Investigation* 1:3–6.

[57] Tzanetakis, G., J. Gao, and P. Steenkiste. 2004. A scalable peer-to-peer system for music information retrieval. *Computer Music Journal* 28:24–33.

[58] Lam, C. K. M., and B. C. Y. Tan. 2001. The Internet is changing the music industry. *Communications of the ACM* 44:62–68.

[59] Leibowitz, N., M. Ripeanu, and A. Wierzbicki. 2003. Deconstructing the KaZaA network. In *Proceedings of the Third IEEE Workshop on Internet Applications (WIAPP 2003)*, San Jose, CA: 112–20.

[60] Liang, J., R. Kumar, and K. W. Ross. 2005. The KaZaA overlay: A measurement study. *Computer Networks Journal* 49(6).

[61] Good, N. S., and A. Krekelberg. 2003. Usability and privacy: A study of KaZaA P2P file-sharing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Fort Lauderdale, FL: 137–44.

[62] Bleul, H., and E. P. Rathgeb. 2005. A simple, efficient and flexible approach to measure multi-protocol peer-to-peer traffic. Paper presented at IEEE International Conference on Networking (ICN'05). Reunion Island.

[63] Lowth, C. 2003. Securing your network against KaZaA. *Linux Journal* 2003(114).

[64] Shin, S., J. Jung, and H. Balakrishnan. 2006. Malware prevalence in the KaZaA file-sharing network. In *Proceedings of the 6th ACM SIGCOMM on Internet Measurement*, Rio De Janeiro, Brazil: 333–38.

[65] Sen, S., and J. Wang. 2004. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Transactions on Networking (TON)* 12:219–32.

[66] Balakrishnan, H., et al. 2003. Looking up data in P2P systems. *Communications of the ACM* 46:43–48.

[67] Liang, J., et al. 2005. Pollution in P2P file sharing systems. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005)*, Miami, FL: 2.

[68] Karagiannis, T., A. Broido, and M. Faloutsos. 2004. Transport layer identification of P2P traffic. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, Taormina, Italy: 121–34.

[69] Spognardi, A., A. Lucarelli, and R. Di Pietro. 2005. A methodology for P2P file-sharing traffic detection. In *Second International Workshop on Hot Topics in Peer-to-Peer Systems (HOT-P2P 2005)*, San Diego, CA: 52–61.

[70] Liang, J., N. Naoumov, and K. W. Ross. 2006. The index poisoning attack in P2P file-sharing systems. Paper presented at Infocom 2006. Barcelon, Spain.

[71] Ripeanu, M. 2001. Peer-to-peer architecture case study: Gnutella network. In *Proceedings of International Conference on Peer-to-Peer Computing*, Linköping, Sweden: 101.

[72] Zeinalipour-Yazti, D. 2002. Exploiting the security weaknesses of the Gnutella protocol. http://www-cs.ucr.edu/ncsyiazti/courses/cs260-2/project/gnotella.pdf, accessed August 23, 2008.

[73] Saroiu, S., P. K. Gummadi, and S. D. Gribble. 2002. A measurement study of peer-to-peer file sharing systems. Paper presented at Proceedings of Multimedia Computing and Networking. San Jose, CA.

[74] Kwok, S. H., and K. Y. Chan. 2004. An enhanced Gnutella P2P protocol: A search perspective. In *18th International Conference on Advanced Information Networking and Applications (AINA 2004)*, Fukuoha, Japan: 1.

[75] Aggarwal, V., et al. 2004. Methodology for estimating network distances of Gnutella neighbors. Paper presented at GI Informatik—Workshop on P2P Systems. Ulm, Germany.

[76] Karagiannis, T., et al. 2004. Is P2P dying or just hiding? Paper presented at IEEE Globecom. Dallas, TX.

[77] Klingberg, T., and R. Manfredi. 2002. *The Gnutella protocol specification v0. 6.* Technical specification.

[78] Matei, R., A. Iamnitchi, and P. Foster. 2002. Mapping the Gnutella network. *IEEE Internet Computing* 6:50–57.

[79] Pouwelse, J. A., et al. 2004. *A measurement study of the BitTorrent peer-to-peer file-sharing system.* Technical Report PDS-2004-007, Delft University of Technology Parallel and Distributed Systems Report Series.

[80] Pouwelse, J. A., et al. 2005. The BitTorrent P2P file-sharing system: Measurements and analysis. Paper presented at International Workshop on Peer-to-Peer Systems (IPTPS). Ithaca, NY.

[81] Yang, W., and N. Abu-Ghazaleh. 2005. GPS: A general peer-to-peer simulator and its use for modeling BitTorrent. In *Proceedings of the International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Atlanta, GA: 425–32.

[82] Bharambe, A. R., C. Herley, and V. N. Padmanabhan. 2006. Analyzing and improving a BitTorrent network's performance mechanisms. Paper presented at Proceedings of IEEE INFOCOM. Barcelona.

[83] Guo, L., et al. 2005. Measurements, analysis, and modeling of BitTorrent-like systems. In *Internet Measurement Conference*, Berkeley, CA: 19–21.

[84] Davis, B. C., and T. Ylonen. 1997. Working group report on Internet/intranet security. In *Proceedings of the Sixth IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Cambridge, MA: 305–8.

[85] Barrett, D. J., R. E. Silverman, and R. G. Byrnes. 2005. *SSH, the secure shell: The definitive guide*. Sebastopol, CA: O'Reilly Media.

[86] Miltchev, S., S. Ioannidis, and A. D. Keromytis. 2002. A study of the relative costs of network security protocols. In *Proceedings of the USENIX Annual Technical Conference, Freenix Track*, Monterey, CA: 41–48.

[87] Poll, E., and A. Schubert. 2007. Verifying an implementation of SSH. Paper presented at Workshop on Issues in the Theory of Security (WITS'07). Braga, Portugal.

[88] Song, D. X., D. Wagner, and X. Tian. 2001. Timing analysis of keystrokes and timing attacks on SSH. In *Proceedings of the 10th Conference on USENIX Security Symposium*, Vol. 10, Washington, DC: 25

[89] Jurjens, J. 2005. Understanding security goals provided by crypto-protocol implementations. In *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM '05)*, Budapest, Hungary: 643–46.

[90] Vaudenay, S. 2005. A *classical introduction to cryptography: Applications for communications security*. New York, NY: Springer.

[91] Longzheng, C., Y. Shengsheng, and Z. Jing-li. 2004. Research and implementation of remote desktop protocol service over SSL VPN. In *Proceedings of the IEEE International Conference on Services Computing (SCC 2004)*, Shanghai, China: 502–5.

[92] Tsai, P. L., C. L. Lei, and W. Y. Wang. 2004. A remote control scheme for ubiquitous personal computing. In *2004 IEEE International Conference on Networking, Sensing and Control*, Taipei, Taiwan: 2.

[93] Miller, K., and M. Pegah. 2007. Virtualization: Virtually at the desktop. In *Proceedings of the 35th Annual ACM SIGUCCS Conference on User Services*, Portland, OR: 255–60.

Part IV

Network-Based Mitigation

In Part IV we will examine several network-based devices that are used to mitigate or detect attacks contained in network traffic. We will not go into detail as to how these devices operate, but instead provide an overview of the common types of devices and their functions. Network-based mitigation methods are only part of an overall security defense system. As their name implies, these devices work best against network-based attacks.

Chapter 12

Common Network Security Devices

In Chapter 12 we will examine three different types of devices. The first, a firewall, is designed to only allow good traffic into a network. The next type, intrusion detection, is designed to examine the network traffic to determine if the traffic is an attack. The last type, data loss prevention, is designed to stop sensitive or private data from leaving a network. All three of these devices are designed to be deployed on the network at or near the connection to the Internet and may be integrated with other network devices like routers.

12.1 Network Firewalls

A network firewall is designed to examine each packet and decide if the packet should be allowed into the network or should be blocked [1–13]. There are several different types of network firewalls based on the layer at which they operate. By this we mean the layer at which the firewall appears to the network (e.g., router or application) and not the layer header information that is used to determine if the packet is blocked. Network firewalls are common in devices like wireless routers. Figure 12.1 shows the general concept behind a network firewall.

Figure 12.1 shows a network firewall with two network interfaces. Every packet that arrives on the inbound interface is compared against a set of rules by the rule engine. If the inbound packet matches the allowed criteria in the rule set, the packet is passed to the internal network. It should be noted that there are many different configurations of firewalls beyond the types discussed here; however, the basic concepts are the same. The rule engine uses the protocol headers and, in some cases, the payload to make the filtering decision.

There are two general types of rules that are used by the rule engine (stateless and stateful). A stateless rule is applied to each packet independent of any other packet. Typically items like port numbers and IP addresses are used in stateless

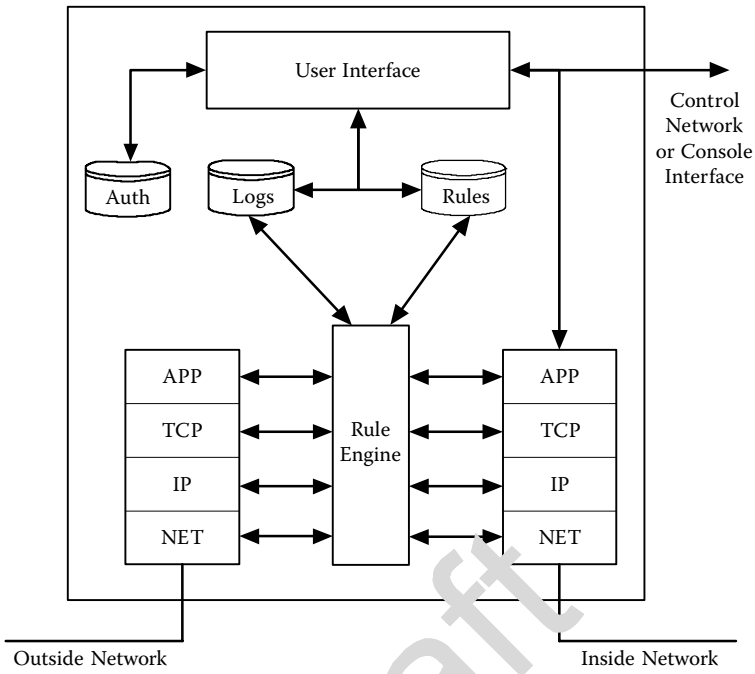


Figure 12.1: Network firewall.

rules. For example, a rule to block all UDP ports but port 53 (DNS) would be a stateless rule. A stateful rule uses multiple packets to determine if the packet should be blocked or allowed. For example, a rule that blocks all incoming DNS packets on port 53 unless there is a pending outbound DNS request pending is a stateful rule. Stateful rules are more complex to implement and configure, but do provide more control over which packets enter the network. Most firewalls implement a combination of both types of rules.

Figure 12.1 also shows a user interface that provides access to the firewall configuration mechanism. The configuration mechanism is accessed through either the inside network, a separate network interface, or a directly connected console. The network-based user interface is often web based and provides a method to update the rules and access log files produced by the firewall. Access to the firewall configuration mechanism through the user interface is often password protected. Most organizations configure their firewalls to be managed from the inside interface. A common mistake users make is to allow their firewall to be managed from the outside interface. Often this occurs with a wireless router. Some organizations

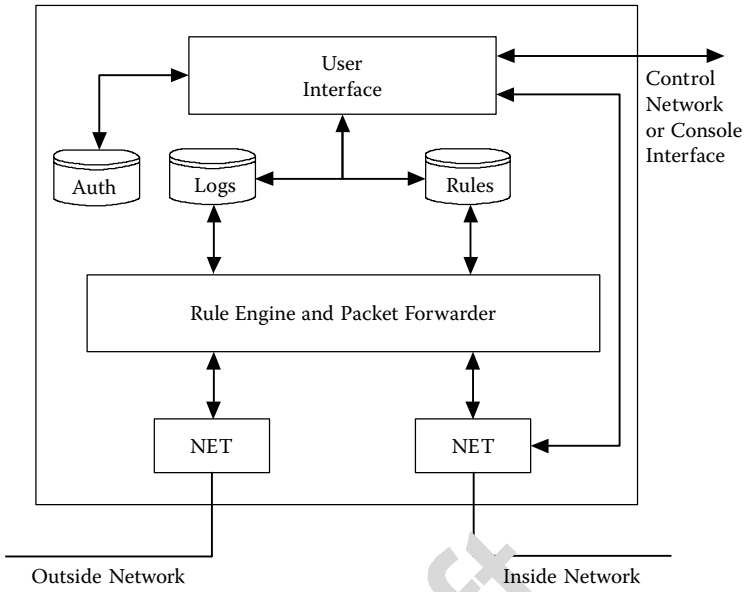


Figure 12.2: Transparent firewall.

will install a separate network that is used to manage all network and security devices. The separate network is typically separated from the Internet by a NAT or firewall.

In this section we will examine four different types of network firewalls. Each of these types are available in the public domain and can be installed on a standard personal computer with two network cards. These firewalls are also available from many different security and network vendors. Figure 12.2 shows a firewall that operates at the physical network layer and is often referred to as a transparent firewall.

A transparent firewall is one that does not appear on the network as a router, NAT, or application. Figure 12.2 shows a firewall with two network interfaces configured to sniff traffic. As far as the network and the devices on the network are concerned, a transparent firewall does not exist on the network. The transparent firewall rules engine examines all parts of the packet to determine if the packet should be allowed based on the rules. If the packet is allowed, it is forwarded to the other interface. Typically a transparent firewall rule set consists of blocking rules and the default case to the forward traffic. The advantage of a transparent firewall is that there is no need to change the network configuration to deploy the firewall. Transparent firewalls could be deployed throughout an organization to help restrict

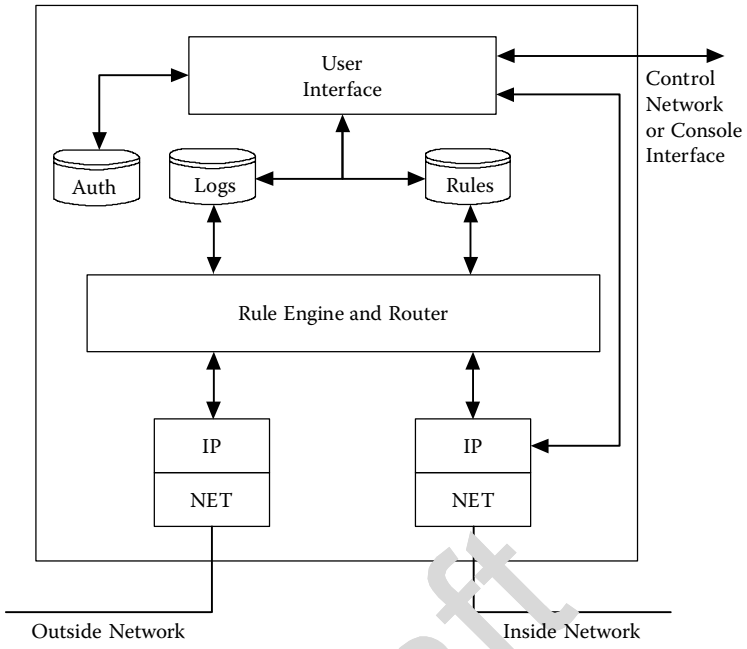


Figure 12.3: Filtering router.

access to internal resources from internal users. A transparent firewall typically implements stateless rules and simple stateful rules because it cannot slow down the traffic flow and does not have much time to process the packets. Another method to implement a transparent firewall is to use one network interface. This type of transparent firewall works on Transmission Control Protocol (TCP) traffic by using TCP reset packets to block unwanted application protocols.

The next type of firewall is often called a filtering or screening router and is shown in Figure 12.3. A filtering router works like a normal router, except it uses a rule engine to determine if the traffic should be filtered. Typically a filtering router allows traffic to pass, and the rule set consists primarily of blocking rules. The rule set is typically stateless since routers are already a traffic bottleneck. Most routers have rule engines that allow them to filter traffic based on protocol type, IP addresses, and port numbers. We discussed this concept earlier in the book when we talked about blocking Internet Control Messaging Protocol (ICMP) echo request traffic as a mitigation technique. A filtering router is often implemented in the router connected to the Internet.

The third type of firewall is embedded into a NAT. The internal configuration is the same as a filtering router, except the rule engine and router are replaced

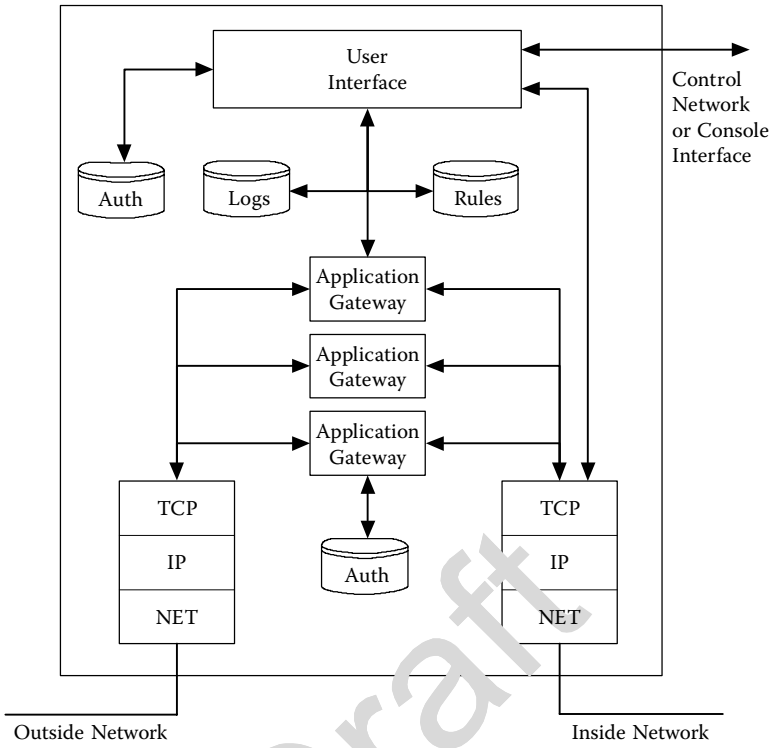


Figure 12.4: Application firewall.

with a rule engine and NAT. As we already discussed, a NAT by default stops all incoming traffic unless there is a tunnel. This makes a fairly effective firewall. The difference between a pure NAT and a combination firewall NAT is that the firewall NAT will use the rule engine to further restrict traffic. A standard NAT uses the IP address and port numbers to determine how to handle the traffic.

The last type of firewall is called an application firewall. Figure 12.4 shows a typical application firewall. An application firewall uses application gateways that allow a user to connect with the gateway running on the firewall, and then uses the application gateway to connect to the internal application. The application gateways may require the user to provide authentication before using the application gateway. The authentication process on the firewall is separate from the authentication process on the final application. An application firewall is very restrictive and is not transparent to the user. A typical application firewall supports tunneling like a NAT in combination with simple firewall rules.

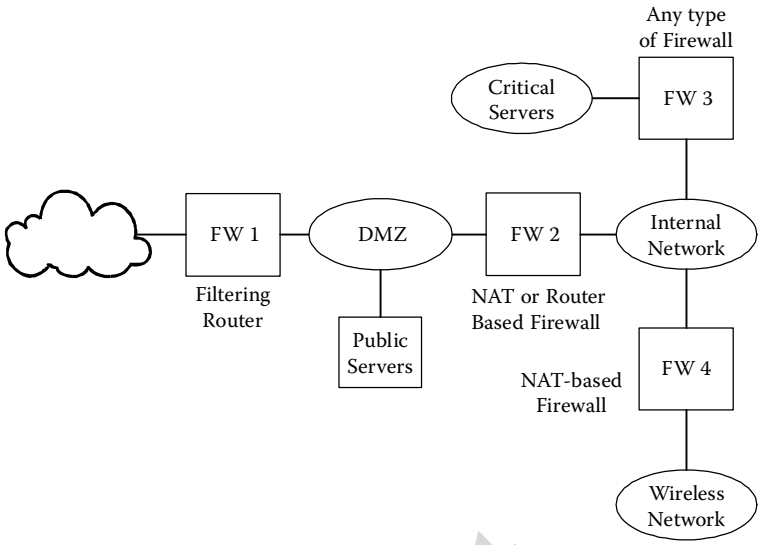


Figure 12.5: Firewall deployment.

One other issue with a firewall is placement within the network. We mentioned earlier that a firewall is typically placed between the organization and the Internet. Figure 12.5 shows several common places to deploy a firewall. It should be noted there are many different topologies that can be used to deploy firewalls, depending on the organization's access and security requirements.

As we see in Figure 12.5, firewall FW1 is a filtering router that is part of the router that connects the organization to the Internet. Behind the filtering router is a network called the DMZ. This network contains servers that are accessed by users in the Internet, like the public web server and email server. The idea behind a DMZ is that users get full access to the network from the Internet, and that the servers placed in the DMZ are under attack. The DMZ is separated from the organization's internal network through another firewall (FW2). This firewall is typically a NAT or router-based firewall. This firewall has more restrictive rules than firewall FW1. Often the only rules in firewall FW2 are to allow inbound traffic from the public servers on a limited number of ports.

Most organizations will implement a DMZ. The figure shows two additional firewalls placed inside the organization. Firewall FW3 is placed to protect critical servers within the organization. Any type of firewall works to protect the internal servers and depends on the type of access that is required for users to access the servers. Firewall FW4 is used to protect the internal network from the wireless network and is implemented as part of the wireless router.

Definitions

Application-based firewall.
A firewall that implements application gateways and often requires user authentication to access the gateway.

DMZ.
A network between two firewalls where public servers are placed.

Filtering router.
A router that uses a rule engine to determine which packets should be routed and which packets should be dropped.

Firewall rule engine.
A process that examines each packet and compares the contents of the packet to a set of rules to determine if the packet should be passed on or dropped.

NAT-based firewall.
A NAT that uses a rule engine to determine which packets to allow or drop.

Stateful rule set.
A set of firewall rules that are applied to each packet based on previous packets.

Stateless rule set.
A set of firewall rules that are applied to each packet independently of any other packet.

Transparent firewall.
A firewall that is transparent to other devices on the network. It operates by sniffing the traffic on the network and passing acceptable traffic on to the other interface.

12.2 Network-Based Intrusion Detection and Prevention

Network-based intrusion detection (IDS) is based on the concept of watching the network for traffic patterns that might indicate an attack. A network-based IDS logs traffic that matches entries in the rule set that indicates an attack. An intrusion prevention device is like an IDS, except it can block traffic that matches an attack rule [14–35]. Just like firewalls, there are several public domain intrusion detection/prevention programs that can be installed on a standard PC platform. Several network security vendors also sell intrusion detection and prevention devices. Figure 12.6 shows a typical intrusion detection/prevention device.

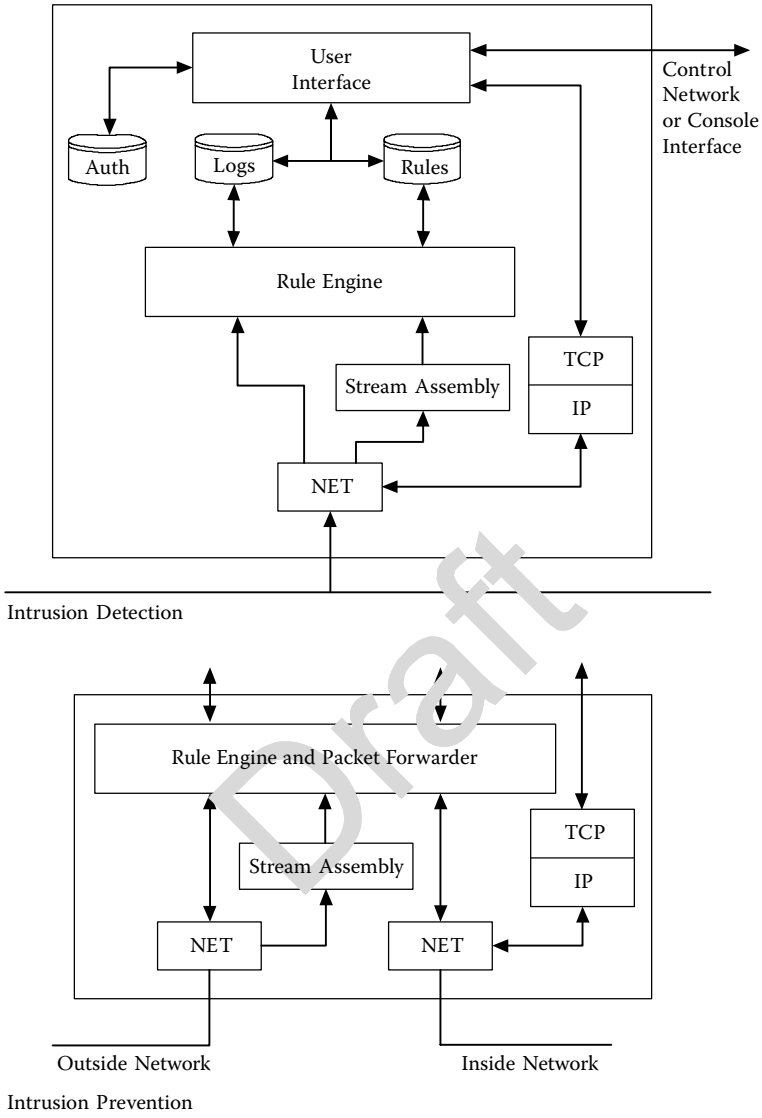


Figure 12.6: Network-based intrusion detection/prevention device.

As we see in Figure 12.6, an intrusion detection device has one network interface that sniffs traffic, which is processed by a rule engine. Since some of the attacks are contained in the TCP payload, the IDS also needs to assemble the TCP stream, which may be contained in multiple packets. The rule engine compares each packet or packet stream to the set of rules to determine if there is an attack.

The rules are divided into two types (signature based and anomaly based). Signature-based uses a set of rules that match data patterns in the packets. For example, a certain string of characters may represent a web-based attack. The rule would consist of the attack string and the port number (80). When the rule engine sees the string in a packet stream destined for port 80, it flags that traffic as a possible attack. With an anomaly-based rule set the rule engine looks for traffic that is not normal for that network. For example, an excess of a certain type of traffic could indicate an attack. The most common type of IDS is signature based, with a few anomaly-based rules.

The primary difference between an IDS and an intrusion prevention system (IPS) is that an IPS typically has two network interfaces and is often configured like a transparent firewall. The IPS uses the rule engine to block traffic that matches the rule set.

There are two issues that make the use of intrusion detection/prevention devices complicated. The first is how well they detect an attack. The rule set of an intrusion detection/prevention device can be complex and will not always correctly identify an attack. There are three possible outcomes from a rule engine. The first is that the rule engine correctly identifies the packet or packet stream as an attack or as normal traffic.

The second outcome is that the rule engine identifies the traffic as an attack when is it not. This is called a false positive. False positives can cause problems by filling up the log files and causing resources (people, time, etc.) to be spent on chasing nonattacks. For intrusion prevention devices false positives can cause the device to block good traffic. This is one reason many organizations do not widely deploy intrusion prevention devices, and if they do, they only enable blocking on a subset of the rules. Another type of false positive is when the device detects an attack, but the attack does not work on any of the devices within the organization. For example, the IDS might detect an attack against the TELNET protocol, but if the organization does not run any TELNET servers, the attack will not affect any devices. This type of false positive fills up the log files.

The third outcome is when the device does not detect an attack. This is called a false negative. False negatives cause obvious problems since the attack traffic goes unnoticed. Manufacturers of intrusion detection and prevention devices work to reduce the number of false positives and the number of false negatives; however, there is a trade-off between the two. Often when you decrease one, you increase the other.

The second issue is the placement of the intrusion detection/prevention device. Referring to Figure 12.7, an IDS/IPS could be deployed in several places. The

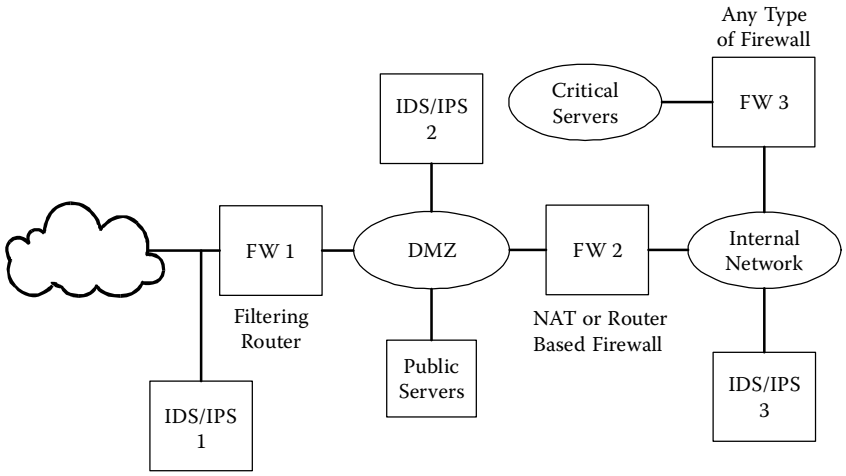


Figure 12.7: IDS/IPS deployment.

first place is between the router and the Internet. Placing an IDS here will show there are a large number of attacks coming from the Internet. Since many of the attacks will not pass through the firewalls, the device logs attacks that do not need to be logged. The only reason to place an intrusion detection device here is to get a sense of the level of attack traffic targeting the organization to see if there are any trends.

Another place to deploy an intrusion detection device would be in the DMZ. This allows you to monitor attacks against the devices inside the DMZ as well as attacks against the internal network. You might also deploy an intrusion prevention device between the internal DMZ firewall and the internal network to stop any attacks that might get by the firewall. The final place to put an intrusion detection device is inside the internal network. This is to detect any attacks that get through the firewall and could be used to detect some internal attacks. An organization might also deploy an intrusion detection device near the critical servers.

Intrusion detection and prevention devices can be useful tools to detect attacks. However, many organizations soon realize these devices can generate a large amount of logs, and unless someone is dedicated to monitoring the devices, their data might be ignored. Another use for intrusion detection is to provide data after an attack. After an attack has been detected on a host, the IDS logs can be examined to see if any network attacks occurred that could have caused the attacks. This information could be used to help reconfigure the defenses to stop the attack in the future or figure out why the defenses failed.

Definitions

False negative.
Marking traffic as good when it really contains an attack.

False positive.
Marking traffic as an attack when it is not.

Intrusion detection.
A device used to detect a network-based attack.

Intrusion prevention.
A device used to detect and block network-based attacks.

12.3 Network-Based Data Loss Prevention

All of the devices we have looked at in this chapter have focused on detecting and stopping attacks from entering the network. A new and growing market is for devices designed to keep confidential and private data (e.g., credit card numbers, social security numbers, and medical records) from leaving an organization. This new market is called data loss prevention (DLP). Figure 12.8 shows a typical configuration of a DLP device.

As we see in the figure, the device looks like an intrusion detection/prevention device. The primary differences are in the rule engine and the addition of proxy servers. The rule engine analyzes the payload of the TCP stream to determine if the content violates the data privacy policy.

There are several methods used to determine if the data is confidential or private. The data can be classified in two types (structured and unstructured). Structured data consists of data elements that can be matched from a list or follow a structured format like credit card numbers. Unstructured data are typically documents that contain private information like memos, letters, or other internal documents. Structured data can be detected using several different methods. Exact matches are where the DLP device has a list of private data elements and compares the network traffic against the list. It should be noted that most of the DLP devices store the hashes of the data elements and compute the hashes of the network traffic and compare the two hashes. Another method to classify structured data is by using regular expressions. Social security numbers often fit into this category since they can be represented in many different forms (e.g., with dashes, without dashes).

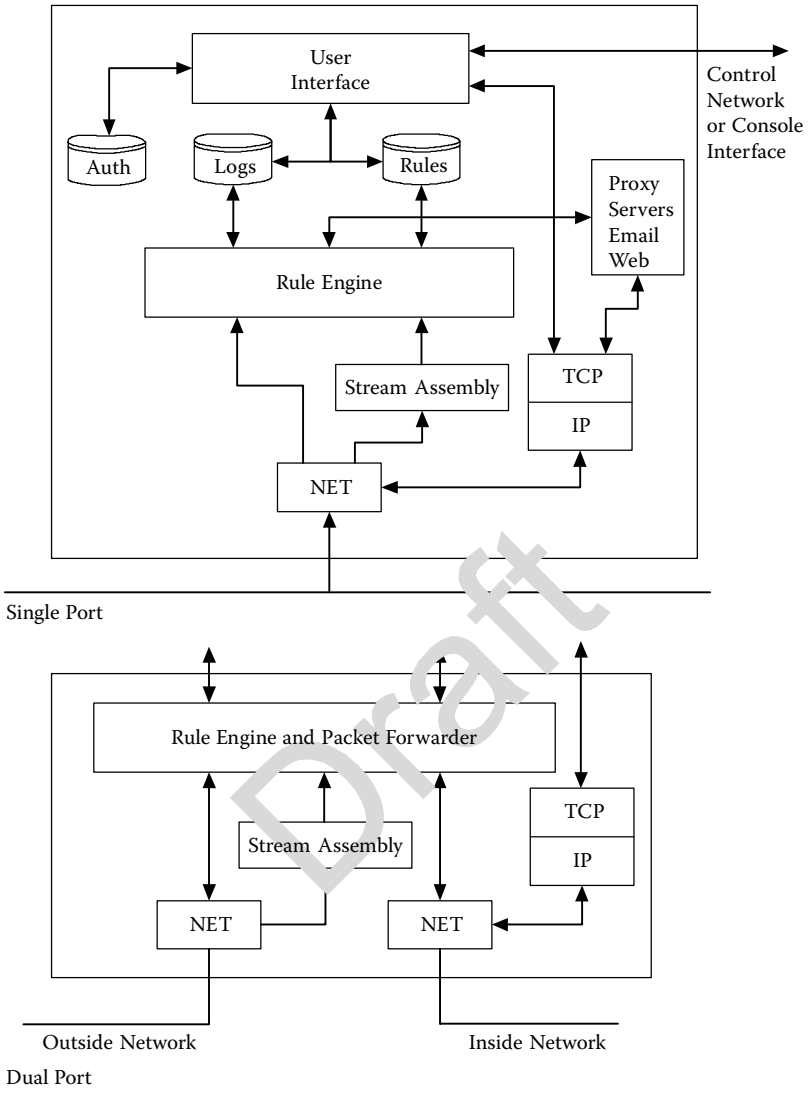


Figure 12.8: Network-based data loss prevention.

A method to handle unstructured documents is called fingerprinting. This method requires that the original documents are analyzed and fingerprints are created. A fingerprint is a hash of part of the document. The network traffic is fingerprinted, and the network fingerprints are compared against the list of fingerprints that were created from the original documents. Another method to handle

unstructured documents is called lexical analysis. This method analyzes the traffic to see if a document matches a set of rules. For example, a medical record might contain medical terms and an id number and something that looks like a patient name. A combination of these items would cause the document to be classified as private.

There are several ways a DLP device handles private data once it detects it. There are some devices that are designed to capture all traffic on the network so an organization can show if there were any violations. These devices typically do not try to stop violations. Another type of device will try to stop violations. This can be difficult since the analysis can take a large amount of time. Many of these devices block violating traffic that is routed through a proxy. The two most common types of proxies are email and web. The email proxy just looks like an MTA. Once the DLP device has detected a violation, it can block it, or in the case of email, it might forward the message to the destination using encryption. This is common for data like social security numbers and credit card numbers. The DLP device may also hold the violating email message until an administrator releases it or deletes it.

Definitions

Data loss prevention (DLP).
A device designed to detect confidential or private data that is leaving a network. The DLP device may also block the violating traffic.

Homework Problems and Lab Experiments

Homework Problems

1. Research different commercial firewalls and public domain firewalls. Determine if there are any differences between the various firewalls. Why would someone use a commercial firewall?

2. Research different commercial IDS/IDP and public domain IDS/IPS. Determine if there are any differences between the various products. Why would someone use a commercial IDS/IPS?

3. Research the Snort IDS rules.

4. Make a case for using an IDS and where you would place the IDS within an organization.
5. Make a case for using an IPS and where you would place the IPS within an organization.
6. Research different methods proposed to create an anomaly-based IDS.
7. Research different data loss prevention (DLP) products. Build a table showing the differences between the various DLP products.

Lab Experiments

1. Use the firewall in the test lab. Try setting up rules to block or allow traffic.
2. Use the Snort IDS in the test lab. Look at the reports to see what types of attacks have been detected. If you have an IDS outside a firewall and one inside a firewall, compare the logs from the two.

References

[1] Lucas, M., A. Singh, and C. Cantrell. 2006. Firewall policies and VPN configurations. Rockland, MA: Syngress Media.

[2] Rowan, T. 2007. Application firewalls: Filling the void. *Network Security* 2007:4–7.

[3] Gouda, M. G., and A. X. Liu. 2007. Structured firewall design. *Computer Networks* 51:1106–20.

[4] Loh, Y. S., et al. 2006. Design and implementation of an XML firewall. In *2006 International Conference on Computational Intelligence and Security*, Guangzhou, China: 2.

[5] Jia, Z., S. Liu, and G. Wang. 2006. Research and design of NIDS based on Linux firewall. In *2006 1st International Symposium on Pervasive Computing and Applications*, Xinjiang, China: 556–60.

[6] Gawish, E. K., et al. 2006. Design and FPGA-implementation of a flexible text search-based spam-stopping firewall. Paper presented at Proceedings of the Twenty-Third National Radio Science Conference, Menout, Egypt. (NRSC 2006).

[7] Goldman, J. E. 2006. Firewall architectures. In *Handbook of information security*, Vol. III, Chapter 170.

[8] Goldman, J. E. 2006. Firewall Basics. In *Handbook of information security*, Vol. III, Chapter 169.

[9] Byrne, P. 2006. Application firewalls in a defense-in-depth design. *Network Security* 2006:9–11.

[10] Hamed, H., and E. Al-Shaer. 2006. Dynamic rule-ordering optimization for high-speed firewall filtering. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, Taipei, Taiwan: 332–42.

[11] Zhou, C., Z. Dai, and L. Jiang. 2007. Research and implementation of complex firewall based on netfilter. *Jisuanji Celiang yu Kongzhi/Computer Measurement and Control* 15:790–91.

[12] Zhang, C. C., M. Winslett, and C. A. Gunter. 2007. On the safety and efficiency of firewall policy deployment. In *IEEE Symposium on Security and Privacy*, Oakland, CA: 33–50.

[13] Firewall, B. I. M. 2006. Product roundup. *Infosecurity Today* 3:12.

[14] Biermann, E., E. Cloete, and L. M. Venter. 2001. A comparison of intrusion detection systems. *Computers and Security* 20:676–83.

[15] Hegazy, I. M., et al. 2005. Evaluating how well agent-based IDS perform. *IEEE Potentials* 24:27–30.

[16] Bace, R., and P. Mell. 2001. *NIST special publication on intrusion detection systems*.

[17] Antonatos, S., et al. 2004. Performance analysis of content matching intrusion detection systems. In *Proceedings of the 2004 International Symposium on Applications and the Internet*, Tokyo, Japan: 208–15.

[18] Jansen, W. A. 2002. Intrusion detection with mobile agents. *Computer Communications* 25:1392–401.

[19] Cavusoglu, H., B. Mishra, and S. Raghunathan. 2005. The value of intrusion detection systems in information technology security architecture. *Information Systems Research* 16:28–46.

[20] Markatos, E. P., et al. 2002. Exclusion-based signature matching for intrusion detection. In *Proceedings of the IASTED International Conference on Communications and Computer Networks (CCN)*, Cambridge, MA: 146–52.

[21] Undercoffer, J., A. Joshi, and J. Pinkston. 2003. Modeling computer attacks: An ontology for intrusion detection. Paper presented at 6th International Symposium on Recent Advances in Intrusion Detection. Pittsburg, PA.

[22] Mell, P., D. Marks, and M. McLarnon. 2000. A denial-of-service resistant intrusion detection architecture. *Computer Networks* 34:641–58.

[23] Pillai, M. M., J. H. P. Eloff, and H. S. Venter. 2004. An approach to implement a network intrusion detection system using genetic algorithms. In *Proceedings of the 2004 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*, Maputo, Mozambigue: 221.

[24] Charitakis, I., K. Anagnostakis, and E. Markatos. 2003. An active traffic splitter architecture for intrusion detection. In *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS 2003)*, Orlando, FL: 238–41.

[25] Axelsson, S. 1999. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, Singapore: 1–7.

[26] Alpcan, T., and T. Basar. 2003. A game theoretic approach to decision and analysis in network intrusion detection. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, Maui, HI: 3.

[27] Sequeira, D. 2003. Intrusion prevention systems: Security's silver bullet? *Business Communications Reviews* 33:36–41.

[28] Rash, M., and A. Orebaugh. 2005. *Intrusion prevention and active response: Deploying network and host IPs.* Syngress. Rockland, MA: Media.

[29] Mattsson, U. 2004. A practical implementation of a real-time intrusion prevention system for commercial enterprise databases. *Data Mining V: Data Mining, Text Mining and Their Business Applications*, 263–72.

[30] Zhang, X., C. Li, and W. Zheng. 2004. Intrusion prevention system design. In *Fourth International Conference on Computer and Information Technology (CIT '04)*, Wuhan, China: 386–90.

[31] Wilander, J., and M. Kamkar. 2002. A comparison of publicly available tools for static intrusion prevention. In *Proceedings of the 7th Nordic Workshop on Secure IT Systems*, Karlstad, Sweden: 68–84.

[32] Janakiraman, R. W., and M. Q. Zhang. 2003. Indra: A peer-to-peer approach to network intrusion detection and prevention. In *Proceedings of the Twelfth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2003)*, Linz, Austria: 226–31.

[33] Ierace, N., C. Urrutia, and R. Bassett. 2005. Intrusion prevention systems. *Ubiquity* 6:2.

[34] Fuchsberger, A. 2005. Intrusion detection systems and intrusion prevention systems. *Information Security Technical Report* 10:134–39.

[35] Schultz, E. 2004. Intrusion prevention. *Computers and Security* 23:265–66.

Appendix A

Cryptology

In this appendix we will examine the basic concepts behind three basic cryptographic methods often used in network security: hash functions, symmetric key encryption, and asymmetric key encryption [1–11]. A hash function is used to convert data into a fixed-length representation of the original data. Encryption is used to convert data into a format that can only be read by someone with secret knowledge. The goal of this appendix is to provide basic information the reader needs to understand the functions of the three concepts, and not the inner workings of the algorithms.

A.1 Hash Functions

A hash function is a one-way function that takes arbitrary length input and converts it to a fixed-length data element. The function is called a many-to-one function, which means there are many different input data sets that produce the same output value. A hash function is designed so that knowing the output cannot give you input. The size of the hash function output determines the number of possible hash values. The typical hash size is 16 bytes, which yields 2 ^ 128 possible hash values (approximately 3.4×10 ^ 38 values).

Hash functions have several uses in the context of network security. Hash functions are a common method for converting passwords into values that are stored in password files. Figure A.1 shows using the hash function to create and check passwords. The hash function is used to create the password that is stored in the password file. When the system needs to authenticate a user, the user provides his or her username (which is used to index the hash function in the password file) and the password. The user password is hashed and the value is compared with the value stored in the password file. Using the hash function allows the system to store the password value in a format that is not easily decoded. Typical methods of decoding the password entry require the use of a software program that takes password combinations and runs them through the hash function and compares the two hashes.

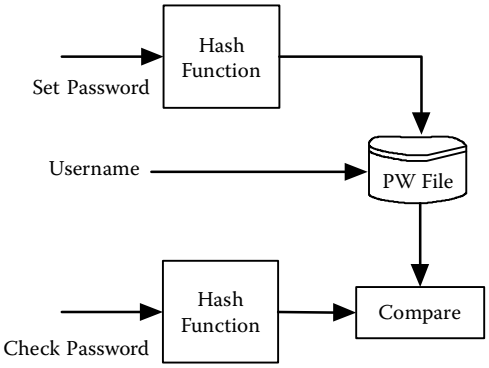


Figure A.1: Using a hash functions for passwords.

Another use of the hash function is to show that data has not been modified, which is called a signature. Hash signatures are also used to uniquely identify a file when issuing a search for files. One issue with hash signatures is how to guarantee the hash signature has not been changed, and that the signature corresponds to the original file. One way is to publish the hash value along with the file. This method is typically used with files that are obtained from web sites. Open-source applications use this method to help ensure the software is valid. Another method to ensure the hash value has not been changed and belongs to a file is to encrypt it. This is called a digital signature and will be discussed in the section on asymmetric encryption.

A.2 Symmetric Key Encryption

Symmetric key encryption is a method where the secret key is known by everyone that needs to encrypt and decrypt the data. As shown in Figure A.2, the encryption and decryption algorithms are the same. When you apply the key to the original data, it is converted to cipher text. The cipher text is converted back to the original data by using the same key that was used to encrypt the data. There are numerous algorithms that provide symmetric key encryption. The differences between the various algorithms are the key size and the computation time of the algorithm.

There are several security-related issues that need to be considered with the use of symmetric key encryption. The first is key distribution. As we saw in Figure A.2, the sender and receiver of the message need to know the encryption

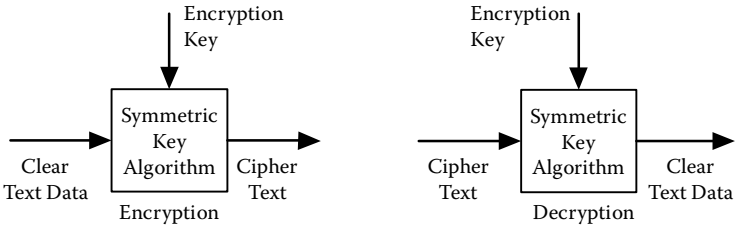


Figure A.2: Symmetric key encryption.

key. The symmetric key is called a shared secret. The strength of any system that uses symmetric key encryption is dependent on the methods used to share and protect the shared secret key. A common use of symmetric key encryption is to encrypt data between two applications. A new secret key is generated for each connection between the applications and is passed between applications using asymmetric key encryption.

Since we sometimes use encryption to prove the identity of an application, device, or person, we need to look at what it means to be able to encrypt and decrypt a message. In symmetric key encryption an encrypted message can only be decrypted by someone who knows the shared secret. Figure A.3 shows an example of using symmetric key encryption to help authenticate users. We see

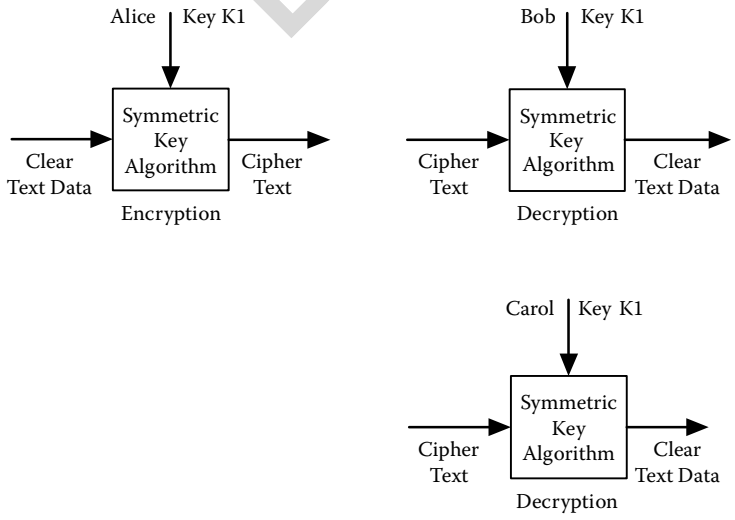


Figure A.3: Multiple key encryption.

that Alice encrypts a message using key K1 and sends that message. If Bob and Carol each know key K1, they can decrypt the message. The question is: What does this prove? Alice knows that only people that know K1 can open the message, and therefore must trust that Bob and Carol keep the key secure. Bob and Carol know that someone who knows key K1 created the message, which could be any of the three users. This shows the fundamental problem with symmetric key encryption. In order to ensure security between any pair of people, we need a separate key for each possible communication. In the example shown in Figure A.3 we would need three keys. This becomes very difficult to manage as the number of users increases. Asymmetric key encryption fixes this problem.

The next issue is the possibility of breaking the encryption. Basically, encryption is a mathematical function that uses the key to manipulate the data. The goal is to make the key large enough that it takes too long to try every possible combination. Unlike passwords, where the key size is short and restricted to printable characters, keys used in typical symmetric key systems are large and are not restricted to certain characters. Typical key sizes for symmetric key systems range from 128 bits ($3.4 \times 10\hat{}38$ possible keys) to 1,024 bits ($1.7 \times 10\hat{}308$ possible keys). This makes trying every possible combination almost impossible. Attackers often try to attack the key generation methods or the key distribution system instead of guessing all possible keys. There are methods to attack the encryption algorithms given enough data. These attacks are beyond the scope of this book.

A.3 Asymmetric Encryption

Asymmetric key encryption, often called public key encryption, uses two keys that are mathematically related. Figure A.4 shows the operation of asymmetric encryption. The figure shows two algorithms, one for encryption and one for decryption. One of the matched keys can be used to encrypt the data, and the other key is used to decrypt the data. The idea is to use one of the matched keys as a public key that is meant to be known by everyone. The other matched key is a private key that is to be kept secret. Figure A.5 shows how the public and private keys can be used to encrypt data between multiple applications. If Alice encrypts the data using her private key, then anyone that knows Alice's public key can read the message. In the figure, both Bob and Carol could decrypt the message, and they know that someone who knows Alice's private key created the message.

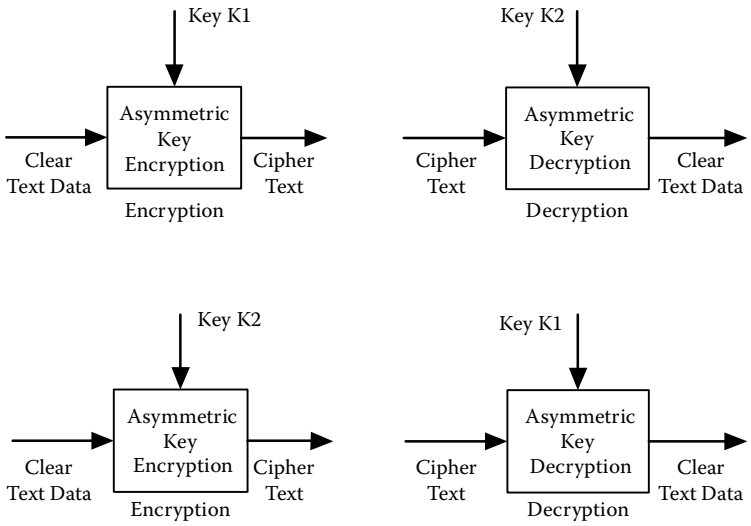


Figure A.4: Asymmetric key encryption.

If Alice wants to send an encrypted message to only Bob, she would encrypt the message using Bob's public key. Only someone that knows Bob's private key could open the message. One very important aspect of asymmetric key encryption is the security of the private key. The private key is often protected by encrypting it with symmetric key encryption. That way, in order for someone to use his private key he would need to know the password.

There are several ways that the asymmetric encryption method can be used. One way is to create a digital signature, as shown in Figure A.6. A digital signature is the hash of the data that has been encrypted using the private key of the sender. This encrypted hash is then sent with the original file. The receiver of the message uses the public key of the sender to open the encrypted hash, and then compares that value with the hash of the received data. If the two are equal, then the receiver knows the data was sent by someone who knows the private key of the sender. A question that comes to mind is: Why not just encrypt the message with the private key instead of encrypting the hash? One reason is that asymmetric encryption is much slower than symmetric key encryption. Another reason is that the goal may be to show that the data has not been changed. Once you decrypt the data, it could be modified. The digital signature can always be used to show the data has not been modified.

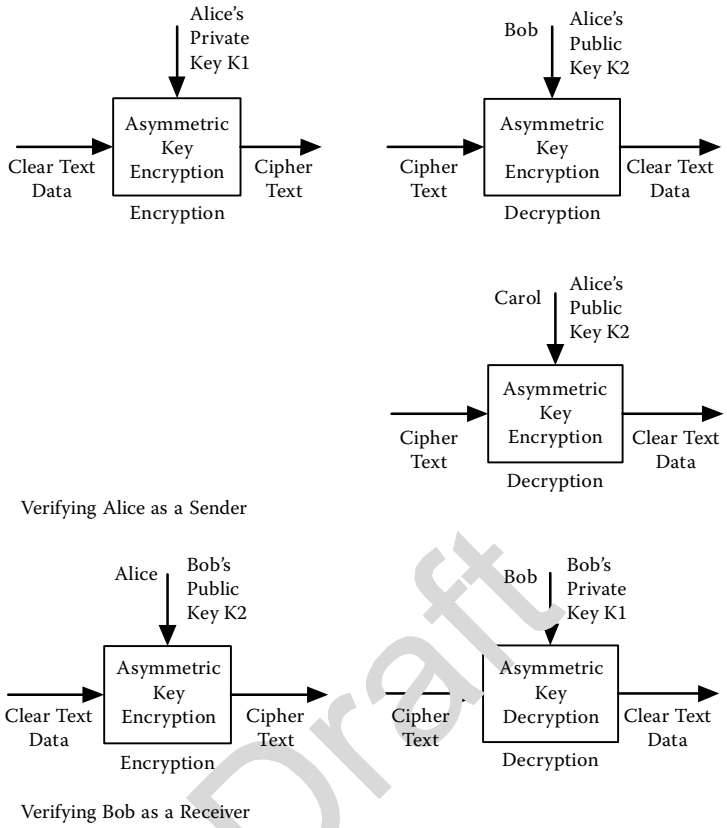


Figure A.5: Using asymmetric key encryption.

Another use of asymmetric encryption is to exchange symmetric keys. Figure A.7 shows an example where symmetric keys are exchanged as part of the message using asymmetrical key encryption. The digital signature is created using Alice's private key and combined with the message. The message is then encrypted using a symmetric key that was randomly generated. The random key is encrypted using the public key of the receiver (Bob). The encrypted key is combined with the encrypted message. When Bob gets the message, he decrypts the symmetric key using his private key and then uses the symmetric key to decrypt the message. The digital signature would then be used to ensure the data was unaltered and was sent by the sender (Alice). In this example, Alice knows that only someone who knows Bob's private key could open the message, and Bob knows that only someone that knows Alice's private key could have sent the message.

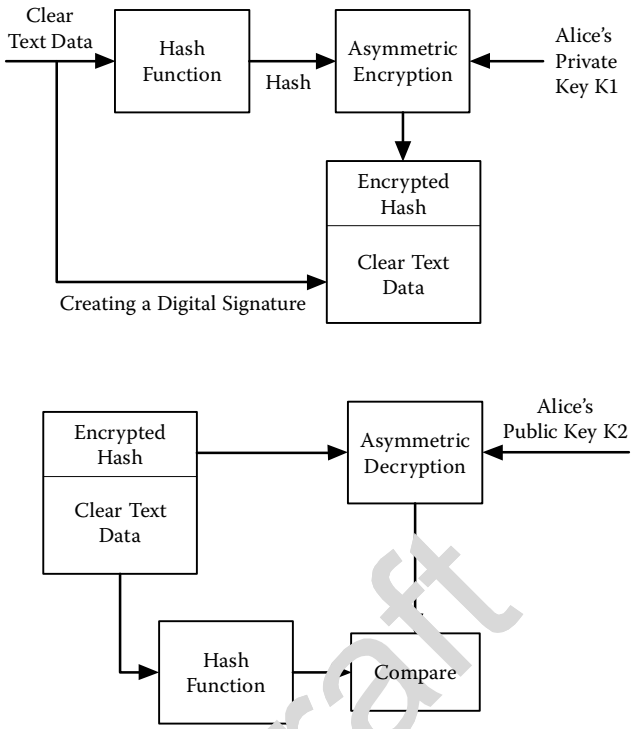


Figure A.6: Digital signature.

Figure A.8 shows how a symmetric key can be exchanged to use to encrypt network traffic. In this case, Alice picks a symmetric key (called a session key) and encrypts it using Bob's public key and sends it to Bob. Bob and Alice can now send data that is encrypted with the shared session key. The session key ensures that only someone who knows Bob's private key can read the network traffic. If Bob wants to make sure it is Alice, he can request that Alice send a message encrypted with her private key.

Definitions

Digital signature.
An encrypted hash of data that can be used to tell if the data has been altered and who sent the data.

Private key.
One half of a key pair that is kept secret by its owner.

Public key.
The other half of the public–private key pair that is known by others.

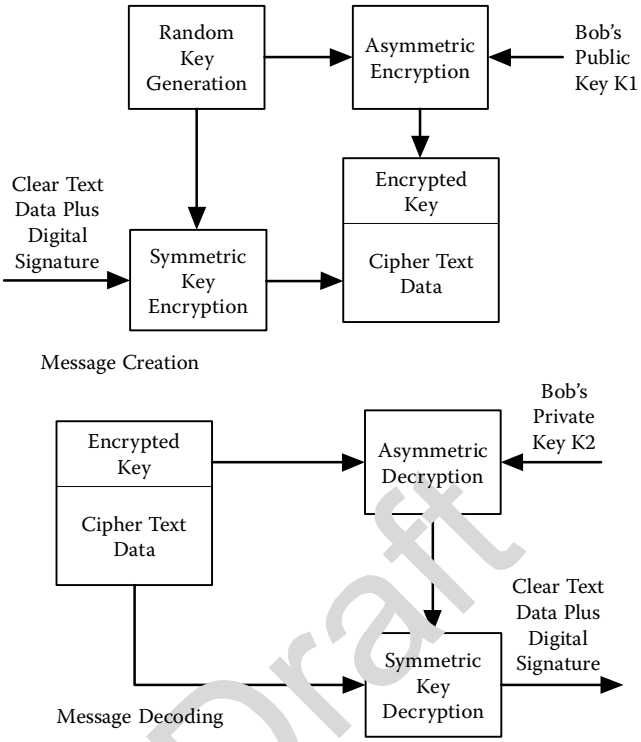


Figure A.7: Message-based symmetric key distribution.

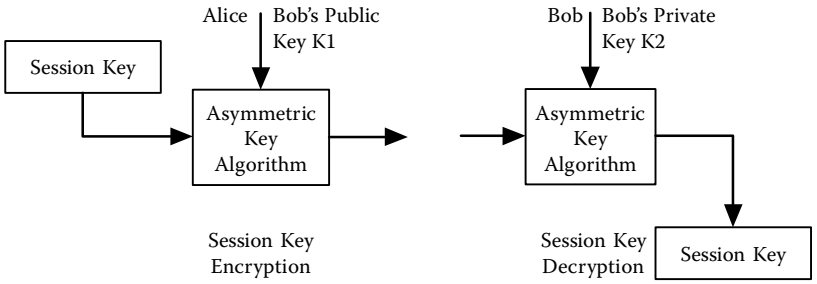


Figure A.8: Network-based symmetric key exchange.

Homework Problems

1. Research the various symmetric key cryptographic algorithms. Build a table comparing the algorithms based on key size.

2. For each of the algorithms in problem 1, assume you can try a key once every microsecond, and, once every nanosecond, compute the time it would take to try every key. How fast would you have to try the keys to break the encryption in a month?

3. Research any weaknesses in commonly used cryptographic algorithms.

4. Research public key infrastructure (PKI) and comment on why there is not a single PKI system.

5. Research tools designed to break password encryption and network encryption (e.g., wireless encryption).

6. Using a web browser, look at the certificates used by the browser and determine the vendors that produced them.

References

[1] Stallings, W. 2006. *Cryptography and network security: Principles and practice*. Englewood Cliffs, NJ: Prentice Hall.

[2] Ferguson, N., and B. Schneier. 2003. *Practical cryptography*. New York: John Wiley & Sons.

[3] Enge, A. 1999. *Elliptic curves and their applications to cryptography: An introduction*. Norwell, MA: Kluwer Academic.

[4] Mollin, R. A. 2001. *An introduction to cryptography*. Boca Raton, FL: CRC Press.

[5] Cohen, H., G. Frey, and R. Avanzi. 2006. *Handbook of elliptic and hyperelliptic curve cryptography*. Boca Raton, FL: CRC Press.

[6] Dent, A. W., and C. J. Mitchell. 2005. *User's guide to cryptography and standards*. Boston: Artech House.

[7] Wayner, P. 2002. *Disappearing cryptography: Information hiding: Steganography and watermarking*. San Francisco, CA: Morgan Kaufmann.

[8] Oppliger, R. 2005. *Contemporary cryptography*. Boston: Artech House.

[9] van Tilborg, H. 2005. *Encyclopedia of cryptography and security*. New York, NY: Springer.

[10] Mollin, R. A. 2003. *RSA and public-key cryptography*. London: Chapman & Hall/CRC.

[11] Boneh, D. 2003. Advances in cryptology-crypto 2003. Paper presented at Proceedings of the 23rd Annual International Cryptology Conference, Santa Barbara, CA, August 17–21.

Appendix B

Laboratory Configuration

This appendix describes a small test laboratory that aids in the understanding of the concepts described throughout the book. The laboratory described is modeled after the laboratory used by the author to teach this topic. The laboratory supports about 100 students. The students access the laboratory remotely, which reduces the space required and the amount of overall computers. The laboratory could be modified to support students sitting in front of the equipment by adding additional computers. The next three sections describe the hardware configuration of the laboratory, the software configuration for the computers in the laboratory, and the issues with remote access. The final section of the appendix provides additional supporting materials that can be used to help with the use of the laboratory. The web site www.dougj.net has additional descriptions of the laboratory configuration and software requirements. The web site also contains scripts and configuration instructions to help set up and run the laboratory.

B.1 Hardware Configuration

The hardware configuration for the laboratory is shown in Figure B.1. This configuration supports the laboratory experiments and programming problems described in the book and will handle 50 to 100 students accessing the laboratory remotely. The hardware requirements of the laboratory are minimal. Figure B.1 shows the laboratory connected to the Internet using a router. It is helpful to have the laboratory on its own subnet; this makes it easier to talk about address ranges, helps keep unwanted traffic from entering the laboratory, and keeps users inside the laboratory from sniffing traffic that is not part of the laboratory. The router can be a commercial router or can be created using a UNIX-based PC with two network cards. Figure B.1 also shows an optional firewall or NAT between the laboratory and the outside network. The firewall/NAT can be combined with the PC-based router. If you want students to remotely access the laboratory, then you do not want to use a NAT. Tunneling through the NAT can be complex. The firewall is

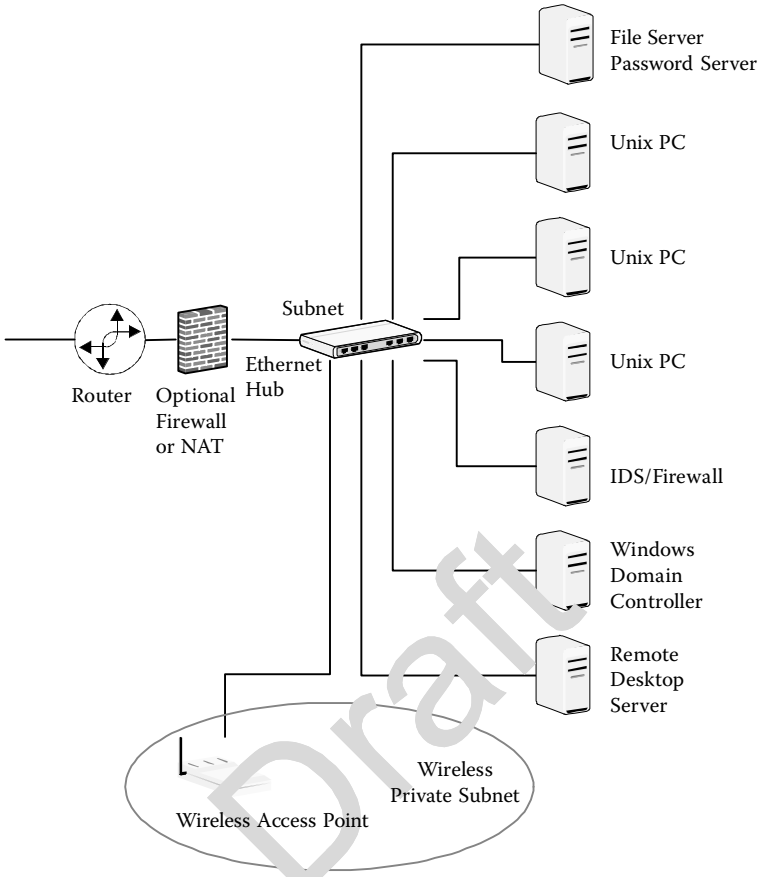


Figure B.1: Laboratory hardware configuration.

used to restrict outbound traffic that comes from network scans or other traffic that should stay off the main network. The computers in the lab are connected with an Ethernet hub. This allows the computer to sniff all of the traffic generated in the network. You can also install a wireless access point to create a second subnet. It is best to make the wireless subnet a private network by configuring the wireless router as a NAT. By then configuring the computers with wireless Ethernet in addition to the wired Ethernet, you can cause traffic to be generated on either network by picking the appropriate subnet.

The UNIX PCs in the lab can be basic computers. The disk requirements are minimal since all of the user file storage is located on the file server. The processor and memory requirements are based on the number of users per machine. The packet sniffer labs consume the most resources. The author created his lab using computers that the department replaced during a lab upgrade. The file server needs

to have enough disk space to handle the number of students. The packet sniffer labs require a large amount of disk space to store the packet captures. Assume 2 gigabytes per user. The windows domain controller can also be a standard PC. The remote desktop computer is the most powerful computer in the lab, and depending on the number of students, you may need to set up a load-balancing server. The author uses two PCs with 2 GB of memory each to the remote desktop computer.

B.2 Software Configuration

The software configuration for the laboratory consists of primarily UNIX-based computers. The author uses FreeBSD. However, any version of UNIX will work. The web site has detailed information about software configurations for the various computers in the laboratory along with supporting programs that are used to create logins and support the packet sniffer programming assignments. The computers need to have the C programming environment installed. The IDS computer runs the Snort public domain IDS. You need to install a web server, an email server (sendmail, IMAP, and POP3), a TELNET server, and a File Transfer Protocol (FTP) server on at least one of the computers. (The same computer does not need to have all of the servers.) You could also install a Domain Name Service (DNS) server and put the addresses of the private wireless network in the server. This would allow students to look at DNS configuration files and sniff DNS traffic. The students do not have login access to the UNIX file server or to the domain controller.

B.3 Remote Access Issues

The author has implemented his version of the laboratory for remote access only. This allows the laboratory to be housed in a small room. The computers used are connected using a KVM so that only two monitors and keyboards are needed. Students use SSH and remote desktop to connect to the laboratory computers. There are some access issues that the author has encountered, so his lab supports both SSH and TELNET access. There have been cases when students have been unable to use SSH and have had to rely on TELNET. A student can use a Windows machine to access the laboratory without installing any additional software if he or she relies on TELNET and FTP. You can also install the X-Windows environment on the UNIX-based machines so students can use a graphical interface. This

requires additional software on the computers the students use to access the laboratory.

B.4 Supporting Material

There are additional supporting materials on the web site. The following tables show the packet formats for TCP/IP and ARP. These tables are helpful for the sniffing programming assignments.

TCP/IP Packet Format (Carried in an Ethernet Packet)

Byte	Field	Comment	
1–6	DA	Ethernet destination address	Ethernet header
7–12	SA	Ethernet source address	Ethernet header
13–14	Type	Type field 0x800—IP 0x806 ARP	Ethernet header
15	Ver/IHL	Ver = 4 IHL = no. of 4-byte words in the header	IP
16	Type	Type of service (typically 0)	IP
17–18	T-len	Total length of packet in bytes, including IP header	IP
19–20	ID	Unique ID for packet	IP
21	Flags	3 bits 0 – DF – MF DF = 0 May fragment MF = 0 last fragment	IP
21–22	Offset	Fragment offset in 64-bit blocks	IP
23	TTL	Time to live no. of hops the packet can live	IP
24	Protocol	Upper-level protocol carried in data	IP
25–26	Checksum	Checksum of the header	IP
27–30	SA	IP source address	IP
31–34	DA	IP destination address	IP
35–36	S-port	Source port	TCP
37–38	D-port	Destination port	TCP
39–42	Seq num	Sequence number	TCP
43–46	ACK	Acknowledgment number	TCP

(*Cont.*)

TCP/IP Packet Format (Carried in an Ethernet Packet) (*Cont.*)

Byte	Field	Comment	
47	len	4 bits; size of the TCP header in 4-byte words	TCP
47–48	Reserved	6 bits not used	TCP
48	Flags	TCP flags U A P R S F U = Urgent, A = ACK, P = PSH, R = Reset, S = SYN, F = FIN	TCP
49–50	Window	TCP window (flow control)	TCP
51–52	Checksum	Checksum for header and data	TCP
53–54	Urgent	Urgent pointer	TCP
55–?	DATA	TCP data	Data
?	FCS	4-byte CRC code	Ethernet header

Common TCP Port Numbers

21	FTP
23	TELNET
25	SMTP
53	DNS
69	TFTP
161	SNMP

ARP Request Packet Format

Byte	Field	Comment	
1–6	DA	Broadcast address (FF:FF:FF:FF:FF:FF)	Ethernet header
7–12	SA	Ethernet source address	Ethernet header
13–14	Type	Type field 0x806 ARP	Ethernet header
15–16	HW type	1 = Ethernet	ARP
17–18	Protocol	Protocol type 0x800	ARP
19	HA length	Hardware address length (6 for Ethernet)	ARP
20	PA length	Length of protocol address (4 for IP)	ARP
21–22	Operation	Operation 1 = ARP request	ARP
23–28	Send HA	Sender hardware address	ARP
29–32	Send PA	Sender protocol address	ARP
33–38	Target HA	Target hardware address 0:0:0:0:0:0	ARP
39–42	Target PA	Target protocol address	ARP
43–60	PAD	Pad bytes	Ethernet
61–64	FCS	Frame check sequence	Ethernet header

ARP Reply Packet Format

Byte	Field	Comment	
1–6	DA	Ethernet destination address	Ethernet header
7–12	SA	Ethernet source address	Ethernet header
13–14	Type	Type field 0x806 ARP	Ethernet header
15–16	HW type	1 = Ethernet	ARP
17–18	Protocol	Protocol type 0x800	ARP
19	HA length	Hardware address length (6 for Ethernet)	ARP
20	PA length	Length of protocol address (4 for IP)	ARP
21–22	Operation	Operation 2 = ARP reply	ARP
23–28	Send HA	Sender hardware address	ARP
29–32	Send PA	Sender protocol address	ARP
33–38	Target HA	Target hardware address	ARP
39–42	Target PA	Target protocol address	ARP
43–60	PAD	Pad bytes	Ethernet
61–64	FCS	Frame check sequence	Ethernet header

Appendix C

Homework Solutions

This appendix contains solutions to selected problems in the book.

Chapter 1

Homework problem 4: Four protocol layers means there are 80 total bytes of header. This leaves 1,420 bytes of payload for each packet. To get the total number of packets transmitted, divide the total payload size by 1,420 and round up. To get the total number of bytes transmitted, take the remainder of the total payload size divided by 1,420 and add 80. This gives you the size of the last packet. Take 1,500 times the number of full packets transmitted and add that to the size of the last packet.

User Payload	Packets	Bytes
1,000	1	1,080
10,000	8	10,640
100,000	71	105,680
1,000,000	705	1,056,400

Homework problem 5:

User Payload	Overhead Bytes	Percent Overhead
1,000	80	8%
10,000	640	6.4%
100,000	5,680	5.7%
1,000,000	56,400	5.6%

Chapter 2

Homework problem 7: The problem with two identical IP addresses is getting the packet to the right computer. The packet will be routed to a network defined by the IP address. If both computers are in the same network, then the hardware address the final router has in its table for the destination IP address will determine which computer gets the packet. What often occurs is that you may connect to one computer one time and connect to the other computer the next time. Often the two computers with the same IP address will detect each other and report an error. If the two computers are not in the same network, then the computer on the network defined by the destination IP address will get the packet.

Homework problem 8: If two computers have the same Ethernet address on the same network, neither will be able to function on the network. Both computers receive the same packet and both computers in turn will respond to the sender. This causes the sending computer to receive multiple responses to the same packet, which causes the protocol to fail. Two computers with the same Ethernet addresses on different networks will not cause any problem, since Ethernet addresses are only used locally.

Lab experiment 2: The vendor code can be used by an administrator to track down a computer on the network. The Ethernet vendor code can narrow down the search. This is useful when two computers end up with the same IP address and the administrator needs to track them down.

Chapter 3

Homework problem 2: There is a list of well-known and assigned port numbers located at www.iana.org/assignments/port-numbers. IANA defines the well-known ports as numbered between 0 and 1,024. The file contains about 10,000 TCP and UDP port numbers.

Homework problem 3: No, these are just the registered port numbers. Applications can use any port they want to.

Homework problem 4: The client application connects, and when the client and server applications try to communicate, they will not be able to since the application protocols are not the same. Some client applications (e.g., TELNET) connect to any server application and allow the user to send data to the application. This can be used to help debug or test server applications.

Homework problem 6: No, an application can use other port numbers than what was assigned. Many server applications can be configured to use other port numbers, and many client applications can be told to connect to a user-defined port number.

Homework problem 9: One reason to spoof a hardware address is to connect to an ISP that expects a predefined hardware address. Most wireless routers, for example, allow the user to change the hardware address. This is called MAC address cloning.

Homework problem 12: No, the Internet is designed to route each packet independently. Routers can be configured to route packets based on network load as well as destination address. There may also be a failure in the current path, and the routers can reroute packets around the failure.

Chapter 4

Homework problem 4: The CVE database can be used to help determine if a computer system has a vulnerability. The database is used as part of an intrusion detection system (IDS) to help the user classify the potential attacks that are seen by the IDS to determine if the attacks can possibly be successful. The database can be used for attacking. First, the attacker determines the version numbers of the operating system and the applications. He or she can then search the database of attacks to determine which attacks can be used against the system.

Homework problem 5: No, not all vulnerabilities can be exploited. The vulnerability may be too complex to exploit, or if the vulnerability is exploited, the damage may be minor. In some cases the fix will alter the operation of the application, and that will need to be weighed against the damage caused by a successful exploit.

Homework problem 6: No, some vulnerabilities are inherent in the design of the application or in the protocol.

Chapter 5

Homework problem 2: The length of the frame is returned by the Ethernet hardware. The code that takes the Ethernet packets will put the packets together to create the packets used by the network layer. The header of the network has a length field to indicate the length of the network layer packet.

Homework problem 3: Technically Ethernet addresses do not need to be globally unique. Ethernet addresses only need to be unique in a network. However, since there is no way to guarantee that two Ethernet cards with the same address are not installed in the same network, they are made to be globally unique.

Homework problem 7: Broadcast packets force every device to read and process the packet they received. This causes unnecessary processing time. In addition, some broadcast packets require that every device respond. This causes excess traffic on the network.

Homework problem 12: The biggest deterrent to using WEP or WPA is key distribution. In a public network it is assumed that users come and go. There needs to be a method to pass the key out to the user. Once you start passing out keys, they are no longer a secret.

Chapter 6

Homework problem 2: The ARP request needs to be a broadcast packet since the requester has no knowledge of who will respond. The response is only useful to the requester, so to help reduce the number of broadcast packets, the response is sent only to the requester.

Homework problem 3: In case the hardware address to IP address mapping changes, the ARP cache needs to automatically expire. This is most common in an environment where IP addresses are dynamically assigned and therefore change.

Homework problem 5:

Host 1

Destination	Next Hop	Interface
129.186.5.0	129.186.5.30	
Default	129.186.5.254	

Router 1

Destination	Next Hop	Interface
129.186.5.0	129.186.5.254	En0
129.186.100.0	129.186.100.254	En1
129.186.4.0	129.186.100.253	En1
Default	129.186.100.252	En1

Host 2

Destination	Next Hop	Interface
129.186.100.0	129.186.100.40	
129.186.5.0	129.186.100.254	
129.186.4.0	129.186.100.253	
Default	129.186.100.252	

Router 2

Destination	Next Hop	Interface
129.186.100.0	129.186.100.252	En0
129.186.5.0	129.186.100.254	En0
129.184.4.0	129.186.100.253	En0
Default	10.0.0.5	En1

Host 3

Destination	Next Hop	Interface
129.186.4.0	129.186.4.133	
Default	129.186.4.254	

Router 3

Destination	Next Hop	Interface
129.186.100.0	129.186.100.253	En0
129.186.4.0	129.186.4.254	En1
129.186.5.0	129.186.100.254	En0
Default	129.186.100.252	En0

Homework problem 10: Printers and other devices that are referenced by IP address are often set up as static. Web, email, and other public servers are also often given static IP addresses.

Homework problem 12:

	Request			Reply		
	Net 1	**Net 2**	**Net 3**	**Net 1**	**Net 2**	**Net 3**
TCP layer						
Source port	5240	NAT	NAT	80	80	80
Destination port	80	80	80	5240	NAT	NAT
IP layer						
SRC IP address	H1	129.186.4.100	129.186.4.100	H3	H3	H3
Dest IP address	H3	H3	H3	H1	129.186.4.100	129.186.4.100

Chapter 7

Homework problem 2: The TCP sequence and acknowledgment numbers are used to count the number of bytes.

Homework problem 3: The TCP and IP layers each add 40 bytes of header to the payload and Ethernet adds 18 bytes. This assumes no options in either TCP or IP headers.

a. 45 bytes

b. 85 bytes

c. 103 bytes

d. 95% of the packet is overhead

Homework problem 5: Assuming that there are no options in the TCP or IP headers, the best size would be 1,500 – 80, or 1,420 bytes.

Chapter 8

Homework problem 1: Local sockets are used to enable processes on a computer to talk to each other. One example is to provide logging of system events. The logging program would act like a server, and any program wishing to log an event would send a message to the logging program using local sockets. Local sockets are designed to look like TCP sockets to simplify the programming. A client is able to communicate using either the local socket or the TCP socket without changing any of the code other than which socket is opened.

Homework problem 4: Yes, there is a limit to the number of open sockets that is allowed. The limit can come from two places. The application can limit the number of connections, thus limiting the number of sockets. The TCP layer has a limit on the number of sockets based on resource limitations. The most common constrained resource is memory.

Chapter 9

Homework problem 1: The protocol exchange is shown in the following table. This assumes 98 bytes for the TCP, IP, and Ethernet headers. The payload consists of the text string and a <cr>. Your solution may vary based on the values of the text strings.

Overhead for part d is 1,764/1,990 = 88.6%.
Overhead for part e is 1,984/1,990 = 99.7%.

Homework problem 4: POP and IMAP are designed to retrieve email from a user mailbox that is considered to be private and is often stored in the user's

Direction	Packet Type	Payload Size	Packet Size
To server	TCP SYN	0	98
To client	TCP SYN+ACK	0	98
To server	TCP ACK	0	98
To client	220 dougj.net + Greeting text	40 (assumption)	138
To server	HELO issl.org	14	112
To client	250 dougj.net Hello issl.org	29	127
To server	MAIL FROM: john@issl.org	25	123
To client	250 john@issl.org Sender OK	28	126
To server	RCPT TO: dougj	15	113
To client	250 dougj Recipient OK	23	121
To server	DATA	5	103
To client	354 Enter Mail	15	113
To server	HELLO	6	104
To server	.	2	100
To client	250 ID Message accepted	24	122
To server	TCP FIN	0	98
To client	TCP FIN+ACK	0	98
To server	TCP ACK	0	98
Totals		226	1,990

directory. Sending email messages does not need authentication since anyone can send email into the email system to be delivered to a user's mailbox.

Homework problem 9: Pros and cons of each type:

SMTP encryption: This could be used to require users to be authenticated before sending email, which could reduce spam. The problem is key distribution. Using public keys will not fix the spam problem and will only mitigate sniffing attacks. Sniffing attacks are better mitigated using other methods.

POP/IMAP encryption: Can be used to stop sniffing, and therefore protect the usernames and passwords. This encryption could also enhance user authentication with additional certificates. The downside is the complexity in key distribution.

User-to-user encryption: This will prevent unauthorized people from reading the email. This can authenticate both the sender and receiver. Key distribution can be complex.

User-to-user encryption would be the best for most secure email.

Homework problem 12: The headers contain the IP addresses and sometimes the host names of each MTA that handled the email. Depending on how the email was sent, there might be information about the email client that sent the message. The usefulness of email tracing is questionable. You can trace the email back to an MTA that was the first to get the email; you might be able to trace back to the IP address of the machine that contacted the MTA. That machine may have changed IP addresses and may have been comprised with a spambot. In general, it is difficult to trace email back to an individual without good log information from the computers used to send the mail.

Chapter 10

Homework problem 1:

a. GET /index.html HTTP/1.1

b. GET /files/index.html HTTP1.1

c. GET /cgi-bin/print-me/?hello%20there

Homework problem 2:

a. Click here for dougj.net

b. Click here for PDF Figure

c.

d.

Homework problem 3:

```
#! /bin/sh
echo Content-type: text/plain
echo

/bin/who
```

Chapter 11

Homework problem 3: The TELNET server is not considered to be secure and should not be used. The only times it is used is for legacy equipment or internal communications. Typically organizations will block the TELNET protocol at the firewall.

Homework problem 4: The TELNET client presents no security risk and is often used to test other protocols and servers.

Homework problem 5: The protocol exchange is shown below. This assumes 98 bytes for the TCP, IP, and Ethernet headers. The payload consists of the text string and a <cr>. Your solution may vary based on the values of the text strings.

Direction	Packet Type	Payload Size	Packet Size
To server	TCP SYN	0	98
To client	TCP SYN+ACK	0	98
To server	TCP ACK	0	98
To client	Greeting text + Username	40 (assumption)	138
To server	Bob (sent as four packets)	4	396 (98*4 + 4)
To client	Bob (bob echoed, 4 packets)	4	396
To client	Password	10	108
To server	Alice (sent as 6 packets)	6	594 (98*6 + 6)
To client	Text to indicate login	40 (assumption)	138
Totals		24	2,046

Overhead is 2,022/2,046 = 98.8%.

Chapter 12

Homework problem 4: An IDS can show you if your systems are being attacked and can alert you if a critical service is being attacked. The best place to put the IDS is in the DMZ, with the rules tuned to match the devices in the organization being protected. An internal IDS can also be useful to see if any attacks are getting through the firewall.

Homework problem 5: An IPS would be best placed inside the organization and set to block certain attacks against the key services. Placing it in the DMZ can work, but you would need to be careful about what attacks you block.

Appendix A

Homework problem 4: A single PKI system has social and political issues that keep it from being adopted. Most people want multiple identities based on what they are trying to do. For example, buying food only requires that your identity is linked to money (you have enough money to pay). Who you are does not matter. There is also the issue of government-sponsored IDS versus local- or business-sponsored IDS.