The design and development of SNIP:
Simple Network Imitator Program

by

Brett Philip Myers

A thesis submitted to the graduate faculty
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Co-Majors: Information Assurance; Computer Engineering

Program of Study Committee:
James Davis, Co-Major Professor
Douglas Jacobson, Co-Major Professor
Cifford Bergman

Iowa State University
Ames, IA
2004

Graduate College

Iowa State University

This is to certify that the master's thesis of

Brett Philip Myers

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

LIST OF FIGURES v

LIST OF TABLES vi

ABSTRACT vii

CHAPTER 1. INTRODUCTION 1

Information Assurance 2

Test Beds 3

What Can Be Done? 4

Objectives 5

CHAPTER 2. RELATED WORK 7

LARIAT 7

ISEAGE 8

Course on Information Warfare 9

CHAPTER 3. BACKGROUND 11

Layered Networking Model and Protocols 11

Libnet 16

Ethereal 17

CHAPTER 4. DESIGN AND IMPLEMENTATION 19

Components 19

Data Structures 21

General Physical, Network, and Transportation Layers Design 22

TCP Functions 24

POP3 Functions 26

POP3 Input 28

Overall POP3 Handling 29

CHAPTER 5. RESULTS 31
Netcat Verification 31
Netscape Mail Verification 32

CHAPTER 6. SUMMARY 37
Compliance with Objectives 37
ISEAGE 38
Future Work 39
Conclusion 40

ACKNOWLEDGMENTS 41
REFERENCES 42

LIST OF FIGURES

Figure 1.1: CERT incidences per year 2
Figure 2.1: The ISEAGE Project 9
Figure 3.1: OSI Model 11
Figure 3.2: Ethernet Frame 12
Figure 3.3: IP Header 14
Figure 3.4: TCP Header 15
Figure 3.5: Libnet protocols 17
Figure 3.6: Ethereal screen shot 18
Figure 4.1: Components 20
Figure 4.2: NetInfo Data Type 21
Figure 4.3: CommInfo data type 22
Figure 4.4: Libnet Build Functions 23
Figure 4.5: TCP Functions 24
Figure 4.6: Three-way connection handshake 25
Figure 4.7: Disconnection handshake 25
Figure 4.8: POP3 Function Prototype 27
Figure 4.9: MailInfo Data Type 28
Figure 4.10: Sequence of function calls 30
Figure 6.1: ISEAGE architecture layout 38

LIST OF TALBES

Table 4.1: POP3 commands 27

Table 5.1: Netcat user generated traffic 33

Table 5.2: SNIP generated traffic for Netcat 34

Table 5.3: Netscape Mail Generated Traffic 35

Table 5.4: SNIP generated traffic for Netscape Mail 36

ABSTRACT

As computer systems become more complex, they become more vulnerable to malicious users. Researchers and professionals are working hard to keep unwanted users out of their networks, but this requires special tools and training for them to be able to adapt to new problems. One requirement for researchers is a good test bed network. This allows them to run experiments and train others on network security without risking live networks. To improve the effectiveness of such networks, tools can be to make these networks seem more realistic.

This thesis proves that a tool can be create that runs on a single machine, but can generate the equivalent traffic of a network, where neither the client nor the server has to exist. However, if a computer on the immediate network was perniciously sniffing traffic, they would not know whether the machine exists or not from the information they gathered from sniffing the network.

This tool has many possible uses and a strong future. It has been developed to be used in Iowa State University's new Internet-Scale Event and Attack Generation Environment (ISEAGE) to provide the background network traffic.

CHAPTER 1. INTRODUCTION

Computers play a vital role in modern society. They manage our information and perform invaluable computations. Computer networking allows us to distribute resources and communicate with others. A network is a group of connected, communicating devices such as computers and printers [4]. These networks fulfill various tasks, communicating through rules known as protocols. Some of the activities governed by these protocols are file transfers, e-mail and viewing web pages, to name a few.

What is currently known as the Internet is based on a research project known as ARPANET sponsored by the Department of Defense Advanced Research Projects Agency (DARPA). Implemented in 1969, this research project connected several computer networks across the nation on a larger network. Currently, the Internet connects thousands of networks throughout the world enabling them to communicate with each other.

With the dependence on computer networks and the Internet increasing, new challenges face the world. A few such problems include individual processor performance, high speed networking, securing communication and data mining algorithm development. Millions of dollars are spent on research and education of many areas relating to computers, especially their networking capabilities.

One such area of study in the computer industry is the study of Computer Security and Information Assurance. The technologies that were meant to improve computer communication and increase work efficiency are being turned against the users they were designed to help by malicious users. Intruders have the ability to gain access to restricted machines and the information that these machines store. Also, malicious programmers can create viruses and worms that quickly and easily spread through the Internet and often times cost companies millions of dollars in losses.

To counteract this, research into Information Assurance has quickly grown to be an area of great concern for both the government and private industry. The remaining sections of this chapter introduce the study of Information Assurance and go on to explain the needs of these researchers and educators. The chapter will conclude by describing the reasons why there is a need of a new tool that will improve research test beds and the requirements for such a tool.

1.1 Information Assurance

The creation of the Internet and the resulting connectivity among computers has caused a security threat to these computers and the information they hold on them. This threat of attack created the need to prepare for and monitor malicious acts. The Computer Emergency Response Team (CERT) Coordination Center has been monitoring incidences since 1988. An incredible increase in the number of attacks occurring each year has occured, from the six incidences reported in 1988 to 137,529 incidences reported in 2003. These incidences can result in financial loss and\or the loss of privacy. Figure 1.1 shows the dramatic increase of incidences over those 15 years [2].

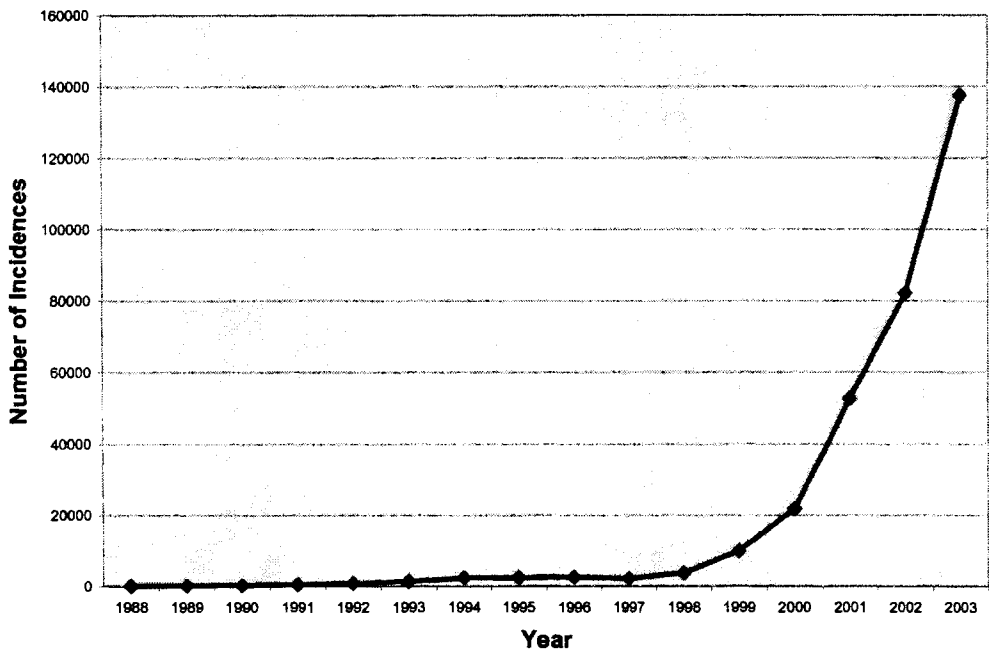

Figure 1.1: CERT incidences per year

As a result, colleges and universities across the United States have steadily increased their research and education programs in the area of Information Assurance. The government is encouraging the growth of these programs in several ways. The National Security Agency (NSA) has created a list of schools that are recognized as Information Assurance Centers of Excellence Institutions. These institutions partner with the NSA to promote the education of

Information Assurance [1]. The National Science Foundation is also encouraging the education of Information Assurance professionals through their Scholarship for Service program. This program provides funding for undergraduate and graduate college students to study Information Assurance and work for the federal government in the field after graduation.

There are several different areas of research in Information Assurance. One area is computer forensics and attack trace back. Finding out the origin of computer attacks is a difficult problem being studied by researchers. This research may require large test bed networks from which to examine the attacks.

Another large area of study is into Intrusion Detection Systems (IDS). Network IDSs monitor network traffic in an attempt to determine if an attack is occurring. This can be quite complicated with the large quantity of traffic that flows through a network. Attacks can be anything from sending a single packet using a buffer overflow to several small packets creating a fragmentation attack. A difficulty in detecting these attacks is that they can be easily camouflaged behind normal traffic and ignored.

One type of attack that IDSs try to detect is denial of service (DOS) attacks. The objective of a DOS is to not allow users access to services that are normally available. DOS attacks can be coordinated to come from multiple attacking machines which can number in the hundreds, or even thousands to create a distributed denial of service (DDOS). Studying these attacks that occur from so many sources can require extremely large test beds.

Another important area of study is in prevention of the distribution of viruses and worms. Viruses and worms have caused companies millions of dollars from the loss of productivity. The observation of how these programs spread across the Internet could be very helpful in preventing them. Once again, a large test bed would be very useful; perhaps even one equivalent to the scale and scope of the Internet.

1.2 Test Beds

Research experiments in the field of Information Assurance should be contained in a protected environment at times. When examining the effects and prevention methods of attacks, worms and other problems computer criminals inflict on computers and networks,

researchers must be sure to protect the outside community. The release of one such experimental computer virus could have a detrimental effect. Researchers need a contained, yet realistic, test bed network in which to perform their experiments that will not harm the outside community.

Another interesting use for test bed networks is educating students. One way to teach people how to defend against attackers is by teaching them the attacker's methodology and showing the students how to penetrate a network. While this is an interesting and effective approach, educators cannot let their students attack live networks. They could damage the network and acquire information they should not access.

Yet, networks setups such that users cannot damage the network are not ideal test situations. For an experiment to be as informational as possible to the researcher, the test bed must be as realistic as possible. Tools need to be created to make these networks more realistic and useful.

1.3 What Can Be done?

One problem when researching and teaching network security is that it is difficult to find a suitable platform in which to conduct tests and teach defense techniques. Since it is often not practical to run some experiments or projects on live networks due to the fact the network could, and at times are expected to, be damaged by the tests. At times, the test bed networks created to run these experiments do not closely enough resemble live networks. The lack of actual network usage can reduce the effectiveness of the test bed to obtain its objective.

A solution to this problem is the creation of a tool that can produce network traffic that is equivalent to the traffic that would be generated by typical users. This would allow researchers and students to see the effects of network traffic on their experiments and exercises.

Such a tool could be deployed on a wireless honeynet. The traffic the tool produces would allow people searching for wireless networks to discover the network, and possibly penetrate the network. Another possible use would be to allow students to sniff traffic and gather information during a penetration test, which is a method used by attackers to gather

information on the network. This tool could also be used to see how new applications and protocols react to typical networks.

1.4 Objectives

The main objective of such a tool is to generate network noise traffic for test beds. The traffic being generated is best called noise traffic, because the traffic produced by such a tool is not actually communicating with anything, but is simply generating realistic looking background traffic, or noise traffic. Computers running this tool are not actually having 2-way communication with other computers, but the communication is done with fictional machines. Because of this, the traffic being generated should not correspond to machines on the network where the research is being done on. If a specific attack is to be tested against a machine, such as a session high jacking, the victim machine should generate its own traffic and open its own connection.

Since traffic itself will not be received by a client or a server application, only machines on the immediate network running applications that are promiscuously sniffing traffic will receive the traffic. However, the computers who are not promiscuously sniffing traffic on the immediate network will react to the traffic the way any computer would react to normal traffic. The traffic that is viewed by the sniffing application should equivalent to any other traffic that would typically be seen on a live network and make the sniffing application have a similar reaction.

The traffic sniffing application must be on the immediate network because the traffic produced by the noise traffic generating application should not be transferred to another network through a router or switch. This will keep the traffic from creating noise on other subnets that could be running other experiments or even possible live networks.
Also, the noise traffic generating application will not pose any security threat to the machine it is running on. If a threat is being tested for, the noise traffic application should not open up new vulnerabilities on its host machine that could alter the results of the experiment.

The last objective of this project is that the code should be easily reused. Since computers and networks are constantly evolving, new protocols may be developed. Different test situations require different network traffic; therefore the implementation must be

flexible. The test noise traffic may not be easily planned for and the application will need to be adaptable to new requirements. Also, parts of this code may need to be incorporated into other applications. Another programmer may be interested in reusing code, perhaps to create a tool to generate traffic to flood a network. By designing the code in components, the code will be more easily transferred to other applications and implementations. One should be able to take different functions from this application and use it in other applications.

Again, the purpose of this thesis is to prove that such a program is possible. The objectives of this thesis can be stated as:

- Produce network noise traffic equivalent to traffic that could be found on the network for test noise
- The noise traffic produced does not transfer to another sub-network
- The application does not create a security threat to the host machine
- Component based for easy code transfer

CHAPTER 2. RELATED WORK

A tool that produces the network traffic equivalent to a computer network is a useful tool for researchers and testers. This has already been addressed at MIT. The approach used at MIT is a very effective method, yet it is not suitable for all situations. The MIT method will be discussed in the first section of this chapter. The proposed tool is a different approach that is required for a new computer security laboratory at Iowa State University, which will be discussed in the second section of this chapter. The final section in this chapter will discuss a class in where this proposed tool will be used to help educate security professionals.

2.1 LARIAT

In 1998 and 1999, DARPA funded MIT to evaluate some network IDSs. To do this, MIT set up test bed networks specifically designed for the tests. The networks were set up to handle offline evaluations of the IDSs being evaluated. Typical network traffic was required to make these tests effective by allowing the IDSs to see normal, noise traffic while they are trying to detect attack traffic [7,8]. An application called Expect was used to generate the traffic, as suggested in [12].

Expect is a tool designed to automate the use of applications, such as Telnet and FTP Clients. Scripts used by Expect can control exactly what the client programs do, such as logging on to servers and downloading files, and when they run [9]. The evaluators set up several client machines to run the scripts that would connect to servers running on the network, which creates the noise traffic on the test bed network.

While this was an effective method to run tests on the IDSs, it was very time consuming. For each test, the network had to be reset. It was then determined that it would be useful to develop a specific evaluation test bed network. Such a test bed would be used to support real-time, automated and quantitative evaluations of IDSs and other Information Assurance technologies [13]. This was the foundation for the Lincoln Adaptable Real-time Information Assurance Testbed (LARIAT).

LARIAT was designed and developed to support real-time evaluations and to create deployable, configurable, easy-to-use test beds [13]. The objectives of the system were to have a test bed network that can be easily setup for evaluations, yet simple enough to use that

the operator would not need to have an understanding of how the system works. The user simply selects the network settings and the attack they wish to run from a database of different attacks.

The noise traffic generation for LARIAT is also done with Expect. An application was developed for LARIAT to allow multiple IP to addresses to come from one computer and to allow users to select different traffic profiles from different Expect scripts. Each profile has different traffic patterns, which statistically copy traffic patterns that have been obtained through observing production networks. The patterns consist of the type of traffic, whether it is POP3 or HTTP or some other protocol, and a scaled amount of each protocol. The traffic being produced by this application is actually connecting to servers that must be set up within the network to complete the communication.

This method is an effective way of generating network noise traffic. However, the objective of the LARIAT tool differs from that of the proposed tool. The LARIAT tool creates traffic that not only serves as noise on the network, but by creating the complete communication process, the servers on the network are actually affected. To some degree, the researchers were interested in the effects on the server, not just creating noise traffic. This meant that not only the traffic generating machine also need to be maintained, but different servers that the traffic will communicate with must be maintained. The proposed tool will only focus on the noise traffic generation and not worry about affecting an actual server, which reduces the number of machines the system administrator will have to maintain. The researchers will only be able to see how the noise traffic affects computers other than the imitated clients and servers that are attached to the network.

2.2 ISEAGE

The Internet-Scale Event and Attack Generation Environment (ISEAGE) is a test bed network being developed by the Information Assurance Center at Iowa State University. ISEAGE will be a test bed network that will create a virtual Internet for the use of testing new network equipment and software. It will also give researchers the chance to examine how interactions occur throughout the Internet, whether it is an attack tracing experiment or watching a worm propagate. Unlike other Internet simulations, the attacks will be carried out

on real equipment [6]. Figure 2.1 shows the different uses for such a test bed, along with some of the control structure.

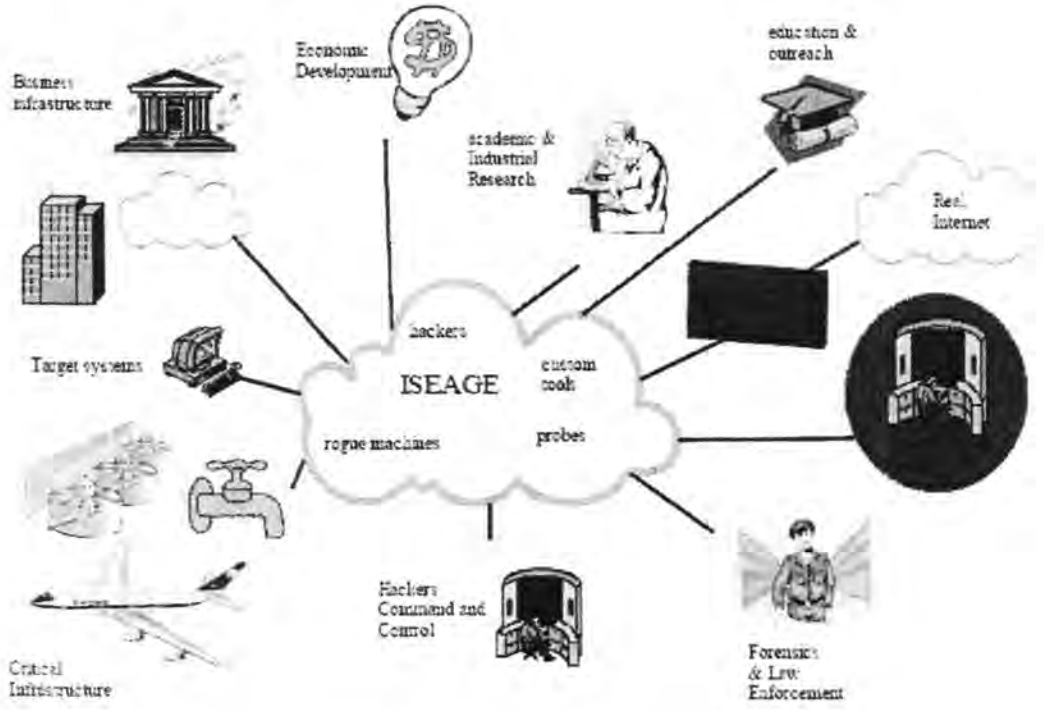

Figure 2.1: The ISEAGE Project

Since ISEAGE is creating a virtual Internet, a comparable amount of noise traffic must be created to make this test bed resemble the Internet, without using all the computers required to create this traffic. The tool being developed in this project will generate the noise traffic required for ISEAGE. This tool reduces the complexity of the system by not requiring actual servers to be running for their interaction

2.3 Course on Information Warfare

One of the more effective ways to train security professionals is to teach them the methodology that attackers use. This allows them to learn about an attacker's thought process and familiarizes them with the tools which they use. Also, it would be helpful to let them use the tools and experience the attacks. By learning about the attacker's tools and

methodology, the students will be more prepare to defend against them. This can be done by allowing the students to attack a test bed network.

This is done at Iowa State in the Information Warfare class offered each spring. Every year, students learn attack methodologies and tools, and then learn how to detect and defend against them. At the end of the class, the students get the opportunity to test their skills on a fake corporate network set up for the class activity, known as 532Corp. This gives the students the experience without endangering any production systems.

CHAPTER 3. BACKGROUND

Computer networking and the Internet have become quite complex. Hundreds of tools and protocols exist and are used regularly. Many of these tools and protocols have become international standards. This chapter details the protocols that are being used in this new tool and the tools being used to test its functionality. The chapter will conclude with a brief discussion of some test bed networks being developed and currently used.

3.1 Layered Networking Model and Protocols

In the late 1970s, the International Standards Organization (ISO) released a standard to cover all aspects of network communication with its Open Systems Interconnection (ISO) model [4]. While this standard is rarely used fully, it standardized the use of the layer framework to allow for communication independent of the architecture it is on. The ISO model called for seven layers: physical, data link, network, transport, session, presentation, and application. These layers were designed so that corresponding layers interact with each other in communicating computers. The layers are then passed to the lower layer as just a data packet, not affecting how the lower layers work. This is represented in Figure 2.1.

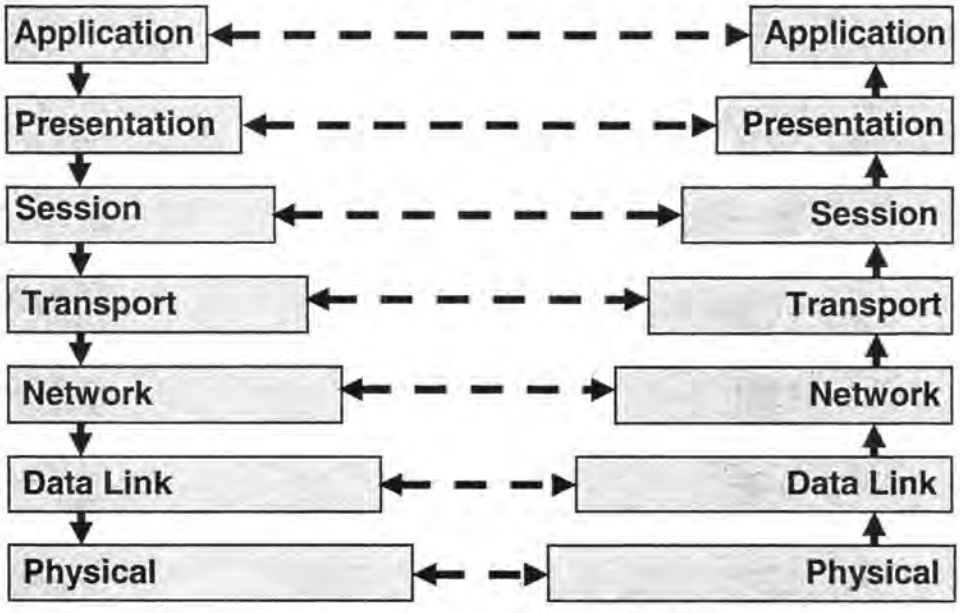

Figure 3.1: OSI Model

This layered approach was also used when developing the TCP/IP Protocol Suite. However, only five layers were used in this suite: the physical, data link, network, transportation and application layers. The physical and data link are defined by the network interface and the physical medium. The application level is defined by the programs that use the application layer. The TCP/IP Protocol Suite has become the international standard that defines the internetworking and transportation layers. Ethernet has also become the standard for the data link layer.

These protocol layers take the information from the previous layer then either add a header to the front of the data or encapsulate the data within a frame. This new information is used to communicate with the corresponding layer on the machine receiving the packet. The Ethernet frame requires the Medium Access Control (MAC) address of both the source and destination machines and the length of the data. The frame also adds seven bytes of alternating '1's and '0's for the preamble and a start frame delimiter (SFD) to signal the start of a new frame. This is the same in all Ethernet packets. However, the MAC Addresses are universally unique to the network interface. The packet format is shown below in Figure 2.2.

Preamble	SFD	Dest. Address	Source Address	Len.
7 btyes	1 byte	6 bytes	6 bytes	2 btyes

Figure 3.2: Ethernet Frame

The Internet Protocol (IP) is the primary standard for the network layer on the Internet. This protocol uses a best-effort approach to delivering its information, meaning there is no error checking or tracking of packets. It uses IP addresses to give each device on the Internet a unique universal address, theoretically. With the limited number of these addresses, network administrators have started using IP Address conserving methods to connect devices to the Internet. IP Addresses currently consist of 4 bytes and are written with a period separating each byte (such as 174.35.185.8).

The IP header, shown in Figure 2.3, requires at least 20 bytes of information, and can hold as much as 60 bytes. The first 4 bits of information give the protocol version (VER). Version 4 (IPv4) is the most used and is the basis of this discussed. There is a push for

version 6 (IPv6) to become the standard, however this is a slow process. The next 4 bits gives the size of the header in number of 4 bit sections (HLEN). The next 8 bits are used for the differentiated services (DS). This is used to state different services that can be used with IP. The next 16 bits give the length of the entire packet in bytes at the IP level (this is the IP header plus the data being carried by the packet).

The next 32 bits deal with possible fragmentation of the packet. At times, the packet might be too large to transport across a network. When this occurs, the packet must be broken into sections and sent separately. This is called fragmentation. The first 16 bits of this section are used to uniquely identify the packet from a particular source and port. The next 3 bits are flag bits. The first of these three flags is reserved and not currently used. The second is set if the packet should not be fragmented, resulting in an error if transmission can not continue. The third flag states whether there are more fragments in the original packet to come. The last 13 bits in this section are the fragmentation offset. This is the number of 8 byte chunks of the original data this section starts on. For example, if a 2,000 byte packet is split into two even fragments, the first fragment offset will be 0 for bytes 0 – 999 and the second fragment offset will be 125 for bytes 1,000 – 1,999. This is the first byte (1,000) divided by 8 to determine the number of 8 byte chunks. These packets must be reassembled at the destination to form the entire original packet.

After the fragmentation section, the next 8 bits hold the time-to-live (TTL) of the packet. Each time a packet is transferred from one network to another, the TTL number is decreased by one. When this number reaches zero, the packet is dropped and does not reach the destination. The next 8 bits state what protocol is contained inside the IP packet (such as TCP or UDP). The next 16 bits are used as the sum check for the packet. This is a number created by a mathematical process done on the packet to check its integrity for error detection. The final required 64 bits are used for the IP address, the first 32 bits are for the source and the second 32 bits are for the destination.

There could be up to 40 bytes of information following the information given for options. These options are not relevant at this time.

VER	HLEN	DS	Total Length	
Identification			Flags	Fragmentation Offset
TTL		Protocol	Header Checksum	
Source IP address				
Destination IP address				
8 bits		8 bits	8 bits	8 bits

Figure 3.3: IP Header

The Transmission Control Protocol (TCP) is a standard protocol at the transport layer level. It establishes a reliable connection between two communicating applications. This protocol is connection oriented since before communication occurs, a port on the client machine must be connected with a port on the server machine through a three-way handshake process. When the communication is completed, the connection must be broken using a four-way handshake process. The TCP protocol is considered reliable because it has error control and packet acknowledgement, so the sending machine knows that its packets have been received. If a packet is ever not received correctly, the sending application can resend the lost packet. This protocol uses port numbers (between 0 and 65,535) to address the different programs using the network interface. Some of these ports are reserved for popular communication tools (such as port 21 for the File Transfer Protocol). The TCP header has two 16 bit port numbers and two 32 bit numbers for the sequence number and the acknowledgement numbers. The sequence number starts as a pseudo-random number generated by the sending machine. This number then has the length of the packet added to it. This new number is the sequence number for next packet sent by the machine. The acknowledgement number is sent by a receiving machine to acknowledge the packet last packet received. This number is the sequence number of the packet received plus its length (or the next sequence number expected).

The header also has 4 bits for the header length (HLEN), 6 bits reserved and 6 flag bits. These flags are for the urgent pointer, an acknowledgement flag, a push flag, a reset flag, a synchronization connection flag, and a finish connection flag. Then there are 16 bit sections for the window size, the checksum, and the urgent pointer. The window size is used to determine the number of packets that are allowed to be sent before acknowledgement is received. The checksum is the same as the IP checksum, used to verify that the packet is intact. Finally, the urgent pointer is used when the data is important and it holds a number that is used to add to the sequence number to know where the last urgent data byte is. The packet is shown in Figure 2.4.

<table>
<tr><td colspan="3">Source Port</td><td colspan="2">Destination Port</td></tr>
<tr><td colspan="5">Sequence Number</td></tr>
<tr><td colspan="5">Acknowledgement Number</td></tr>
<tr><td>HLEN</td><td>Reserved</td><td>Flags</td><td colspan="2">Window Size</td></tr>
<tr><td colspan="3">Checksum</td><td colspan="2">Urgent Pointer</td></tr>
<tr><td>8 bits</td><td colspan="2">8 bits</td><td>8 bits</td><td>8 bits</td></tr>
</table>

Figure 3.4: TCP Header

The data carried by the transport layer comes from the application layer. There are hundreds of different application layer protocols available. One of which is the Post Office Protocol version 3 (or POP3), which allows client machines to contact e-mail servers and receive the users e-mail. This is a relatively simple protocol, only having nine commands. These commands are USER, PASS, LIST, STAT, RETR, TOP, DELE, RSET, UIDL, APOP, NOOP and QUIT. The USER command is followed by the user's login name. The PASS command is followed by the password that username. The STAT command requests the number of e-mails available for this account and their size. The LIST command requests the size of each e-mail available. RETR is followed by the message number to request the e-mail message. TOP is followed by the message number and the number of lines requested. The server will reply with the header for the message and the number of lines requested by

the client. DELE is followed by an e-mail message number and marks that message for deletion. RSET removes all marks made by the server. UIDL is followed by a message number and returns the unique-identifier listing for the message, which gives each e-mail ever received by a users account a unique character string of one to 70 character in the range of 0x21 to 0x7E [10]. By using this number, e-mail clients will be able to recognize if they have already downloaded the message. The e-mail number cannot be used for this purpose because that number could be changed after the client has logged off the server. The APOP command is followed by user's login name and an MD5 sum check to the timestamp sent by the server appended with a secret shared between the client and the server, giving another level of authentication and security for the server. The NOOP command does not have any arguments with it and the server just replies with it status. QUIT deletes all the messages marked for deletion and logs the client out of the mail server.

3.2 Libnet

The Libnet Library is a packet creation and injection library developed in C by Mike Schiffman. It focuses on the TCP/IP Protocol Suite and handles several of the corresponding protocols, which are shown in Figure 2.5. The complete packet is generated in a data buffer defined by libnet called libnet_t, which holds the different headers, packet information and data held by the packets.

The library supports packet injection in two ways. It allows the application programmer to open a raw socket interface to allow the system to handle its own data link layer or through link-layer interface. The link-layer interface allows the application programmer to control the data link protocol and change the MAC address of the packet.

Libnet is licensed under the BSD license agreement.

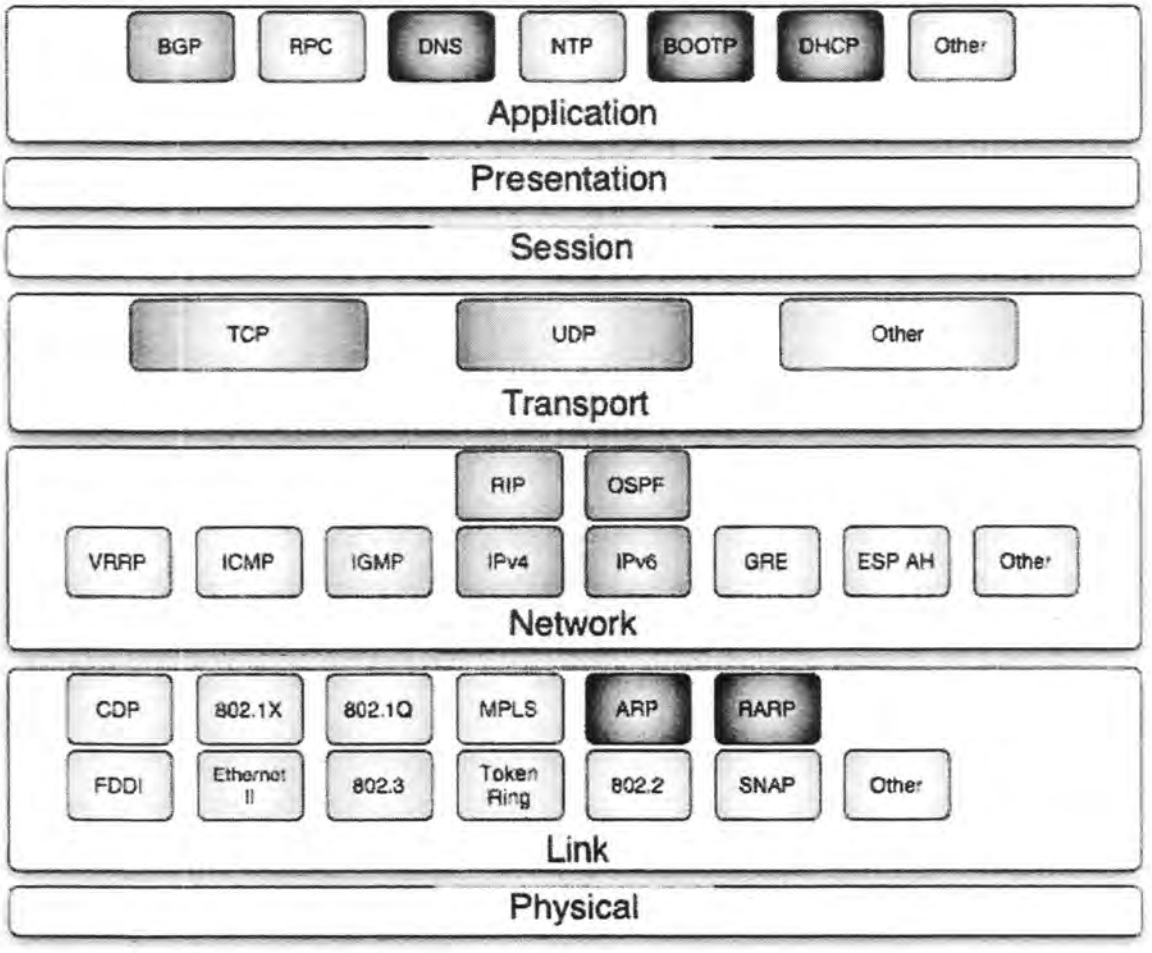

Figure 3.5: Libnet protocols [15]

3.3 Ethereal

Ethereal is a popular open source network traffic analyzer. It is used to take network traffic and examine the packets to see what they are. Currently, 472 different protocols can be dissected. Included in these 472 protocols are IP, TCP, Ethernet, and POP3. When dissected, Ethereal can differentiate the information in each section of the packet. A screen shot of Ethereal is shown in Figure 2.6.

It can be used with TCPDump, which is a network sniffer. Using these tools together, attackers can gather information about the network and researchers can watch traffic patterns.

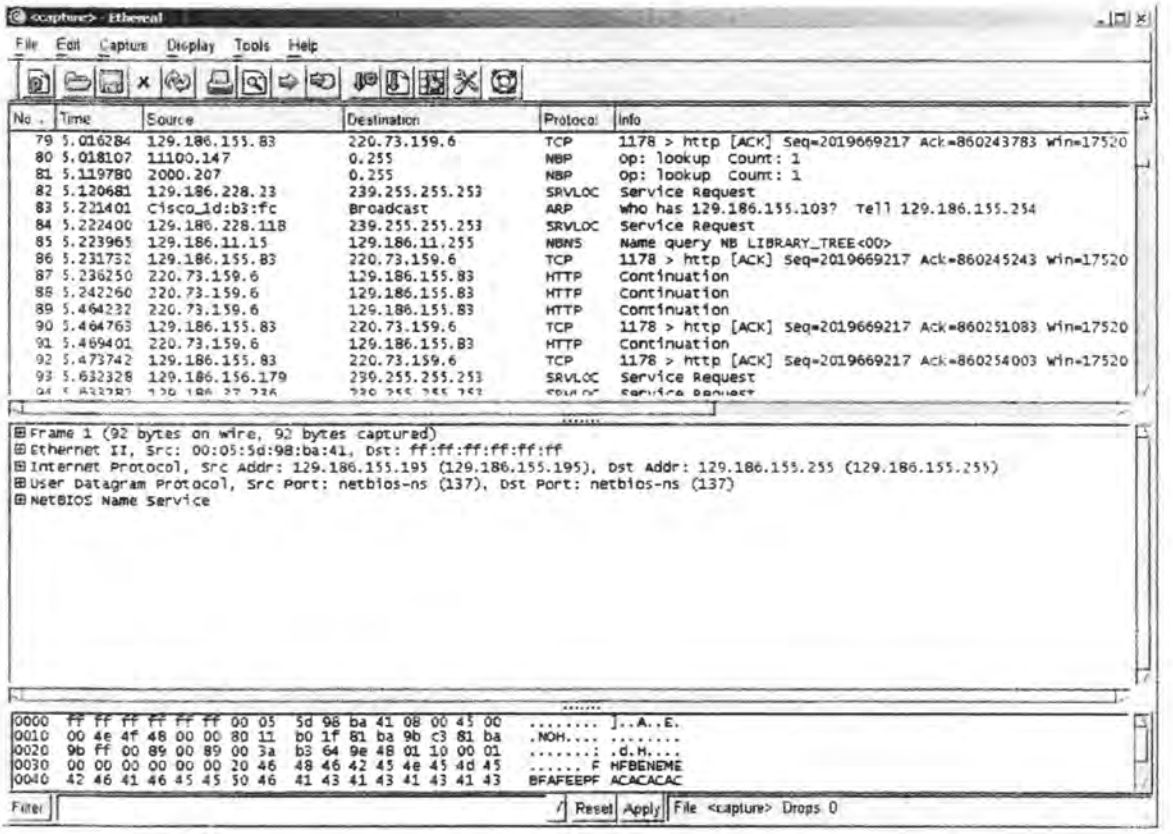

Figure 3.6: Ethereal screen shot

CHAPTER 4. DESIGN AND IMPLEMENTATION

The design of this tool is based on the layered model of computer network communications. Each layer of the model has associated protocols, and each of these protocols has its own set of operational processes. These processes are needed for the setup, interaction, and disconnection of a communication process with the protocol and will have corresponding functions. Each function will handle the necessary events required by the particular protocol and the functions will communicate with other functions for different protocols, similar to the layers of the network model do, to complete communication. Data structures have been defined to assist with the interactions of these functions. Overarching functions have also been created to generate the complete communication process between two computers at the application layer. The rest of this chapter will discuss the design of these different functions and the data types used within them.

4.1 Components

To make this code easy to reuse, it has been created in components. This way the different components can be reused, like the layered networking model. A visual representation of this is given in Figure 4.1. By using components, different functions will have their own function set, represented by the objects in Figure 4.1. Each protocol will have at least one function that creates the packet information. This is all that some protocols, such as IP, require. Other protocols require different series of packets to create connections and communicate appropriately, such as TCP. These procedures are grouped together into control functions.

However, these components are dependent on other components. IP can not simply write a random IP packet to the wire without having a purpose. Also, a TCP packet requires the IP packet to be sent.

Yet, by creating these components, it makes it easy to use the TCP code with multiple application layer implementations. The POP3 function will send data to the TCP function to be transferred. Later, an implementation of the File Transfer Protocol (FTP) could be created, and use the same TCP functions developed for the POP3 implementation.

It is also possible to implement other transportation layer protocols, such as User Datagram Protocol (UDP). Then the UDP functions could use the already existing IP functions.

All of these components will communicate with each other through new data structures created for them, which are discussed in the next section. But to make the components fit together; they must both use the same data structures.

By dividing this project into different components, these components can be reused with different protocols once they are implemented. If a researcher needs to use FTP to complete an experiment, then the researcher can reuse the TCP and IP components developed for this project in the new implementation. Another possibility is that a researcher may not want a complete communication process, the components developed in this project to generate the packets to be sent could be reused, and a new main control function to create the incomplete traffic could be made.

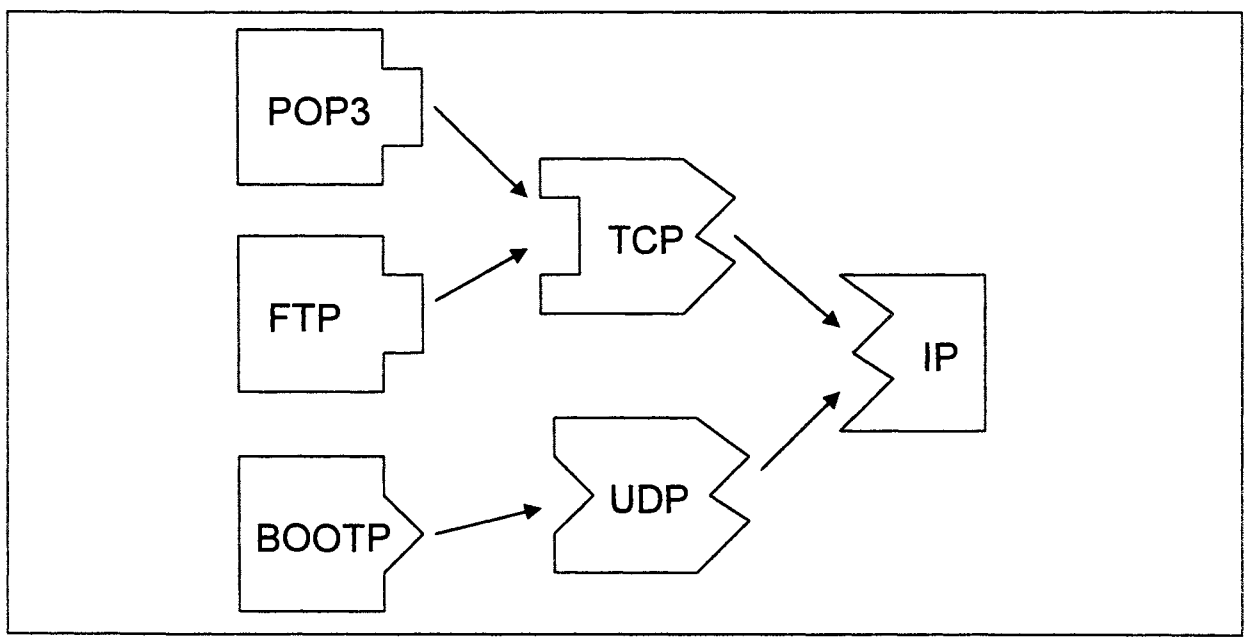

Figure 4.1: Components

4.2 Data Structures

General data structures have been created to simplify the communication between functions. Two structures have been created and are used by the TCP and IP functions. Other applications that use the functions for the TCP and IP protocols will also use these structures. These structures are defined as data types called NetInfo and CommInfo and are described below.

Each computer being represented will have a NetInfo data type shown below in Figure 4.2. This data structure will hold the represented computer's IP address, port number being used, MAC address, IP identification number and TCP sequence number for the communication.

```
typedef struct NetworkInformation
 {
  u_long IP;
  u_short Port;
  u_char EthAdd[6];
  u_int32 Seq;
  u_short ID;
 } NetInfo;
```

Figure 4.2: NetInfo Data Type

Another data type is needed to hold the information passed by the imitated communication. The CommInfo data type, shown in Figure 4.3, was created for this purpose. The structure has two NetInfo data types for the machines that are participating in the communication, a character pointer, a data length variable, and a libnet_t buffer.

The two NetInfo data types are there to make the correlation of the two machines' communications easier, such as the sequence numbers and the acknowledgements, if needed. The NetInfo data types are not pointers since two CommInfo data types cannot share a NetInfo data type because each communication interaction requires sequence numbers and identification numbers to be unique to that connection. This data structure simplifies handling one server and multiple clients. This is because a single server can be represented

in multiple CommInfo, with different sequence numbers and identification numbers, but the key information, such as the addresses, must be copied into the data structure.

The data for the application layer is generated by the individual Application Layer Functions. The function passes this data to the Network Layer through the character pointer and data length variable. Finally, the libnet_t buffer is required for the libnet functions that are being used within the tool.

```
typedef struct CommunicatInformation
 {
  NetInfo Server;
  NetInfo Client;
  char *data;
  int DataSize;
  libnet_t PacketBuff;
 } CommInfo;
```

Figure 4.3: CommInfo data type

4.3 General Physical, Network and Transportation Layers Design

The physical and network layer communications are handled using the libnet library. The libnet library includes functions to build the IP header, the TCP header, and the Ethernet frame for the tool. These functions are independent from one another. They create the respective packet headers and add them to the packet buffer. The data buffer for the packet information is the libnet data type libnet_t, which holds all the information needed to write the packet to the network. The functions require all the particular header information and the libnet packet data type (libnet_t) that the packet is being generated in. The function prototypes are listed in Figure 4.4.

```
libnet_ptag_t libnet_build_tcp ( u_int16_t sp, u_int16_t dp, u_int32_t seq,
  u_int32_t ack, u_int8_t control, u_int16_t win, u_int16_t sum,
  u_int16_t urg, u_int16_t len, u_int8_t * payload, u_int32_t payload_s,
  libnet_t * l, libnet_ptag_t ptag )

libnet_ptag_t libnet_build_ipv4 ( u_int16_t len, u_int8_t tos, u_int16_t id,
  u_int16_t frag, u_int8_t ttl, u_int8_t prot, u_int16_t sum, u_int32_t src,
  u_int32_t dst, u_int8_t * payload, u_int32_t payload_s, libnet_t * l,
  libnet_ptag_t ptag )

libnet_ptag_t libnet_build_ethernet ( u_int8_t * dst, u_int8_t * src,
  u_int16_t type, u_int8_t * payload, u_int32_t payload_s, libnet_t * l,
  libnet_ptag_t ptag )
```

Figure 4.4: Libnet Build Functions

The IP addresses and the MAC addresses are part of the input to the application. These addresses may be tied to particular users, if necessary. This depends on the particular application layer protocol being used. Also, the MAC address should not correspond to any MAC address on the local network. By making sure that the MAC address does not exist on the local network it assures that for the packet will be dropped and not transferred to another network. If the MAC address does exist on the immediate network, the network device with that MAC address will attempt to handle the incoming traffic, possibly harming that machine. This way, the only computers to read the packet off the network are computers in promiscuous mode the read everything off the network.

In the IP header, the check sum is generated by the libnet function. All other pieces of the header are defined by the user, including fragmentation, time-to-live and the packet identification. Libnet also allows future implementations to be able to fragment their packets. Also, the packet identification number is pseudo-randomly generated for the TCP connection's initial value. It is modified throughout the TCP connection.

In the TCP header, the sequence numbers are generated pseudo-randomly by a function in the libnet library, libnet_seed_prand. The check sum for the TCP packet is also generated by the libnet function. The rest of the packet is defined by the programmer, such as the window size. The identification number and sequence number are pseudo-random for

their initial values to give the packets authenticity, since these numbers are pseudo-randomly generated for real packets.

Libnet also has functions to initiate the network interface, write the packet to the network and deactivate the network interface. There are two options for opening the network interface, the raw socket interface or the link layer interface. The link layer interface uses the computer's actual MAC address and creates the Ethernet header. The raw socket interface allows the programmer to manipulate the Ethernet header, this option was chosen so that the MAC address could be modified.

4.4 TCP Functions

There are three different processes in TCP communication: three-way handshaking for connection, the regular communication process and the four-way handshaking for connection termination. Each of these processes have their own functions, and their prototypes are listed below in Figure 4.5. The connection and disconnection handshakes only require the two computers that are being represented. The communication function will update the computer's information when needed due to the communications events. This is also where the options for the TCP and IP headers will be set. A diagram of the sequence of the packets sent for the three-way connection handshake and the disconnection handshake are shown in Figure 4.6 and 4.7, respectively.

```
int TCPConnect ( CommInfo *Comps )
int TCPDisconnect ( CommInfo *Comps )
int TCPSendPacket ( CommInfo *Comps, int src, int des, u_int8_t Ops )
```

Figure 4.5: TCP Functions

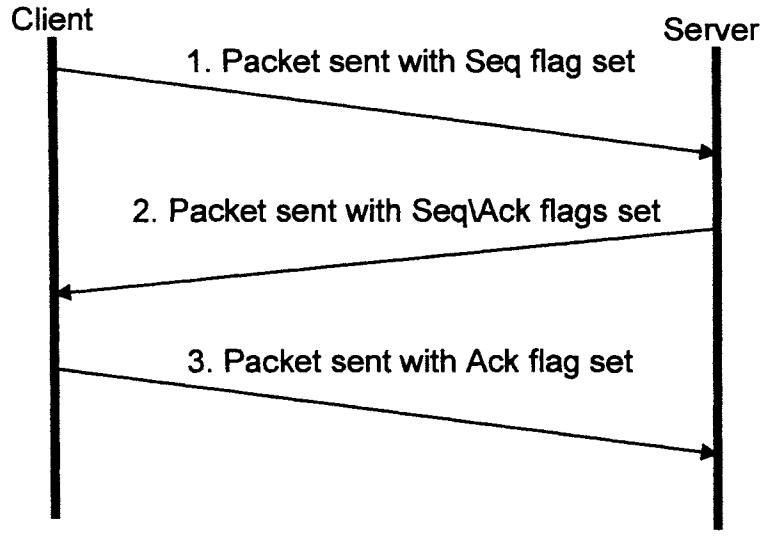

Figure 4.6: Three-way connection handshake

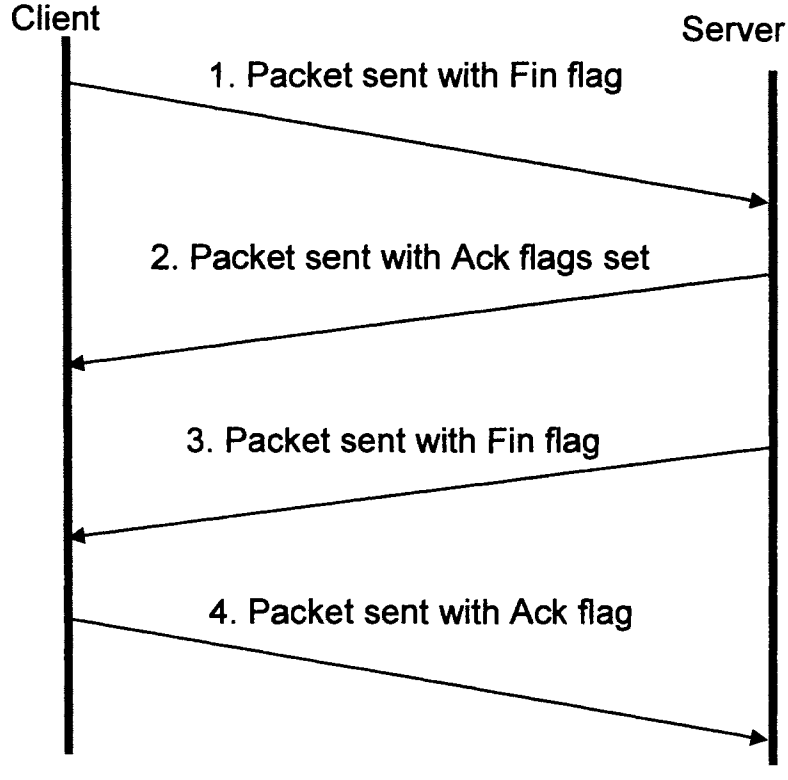

Figure 4.7: Disconnection handshake

4.5 POP3 Functions

Because the libnet library does not have functions to form the required application level protocols, they had to be made. This implementation is imitating POP3 communication, so only one specific function has been created. The function, whose prototype is listed below in Figure 4.8, requires the following information:

- A pointer to the computers being imitated
- The command to be sent (shown in Table 4.1)
- The message number, if one is being manipulated
- The number of lines requested, if the TOP command has been called
- The mailbox information
- The user identifier that is requesting information

A new data structure has been created for the POP3 function. This data structure is defined as a data type called MailInfo, which is shown in Figure 4.9. This data structure holds the information listed below.

- Mail server name
- Mail server information
- Number of users
- The list of user names
- The list of passwords
- The MD5 sum check for authentication
- The number of e-mails for each user
- The e-mails stored for each user
- Unique-id listing number for each e-mail
- The corresponding size of each e-mail
- Each user's computer information

With the application layer protocol function created, several higher level functions are created to control the interactions between the client and the server. For this implementation, one function will handle all the commands in Table 4.1. This function will not only handle

the application layer interaction, but also call the necessary methods for generating the TCP and IP headers and finally writing of the packet to the network. Also, for this implementation, each user will be tied to one specific computer and there will only be one server.

Table 4.1: POP3 commands

POP3 Command	Typedef	Number
USER	POP3_USER	1
PASS	POP3_PASS	2
STAT	POP3_STAT	3
LIST	POP3_LIST	4
RETR	POP3_RETR	5
TOP	POP3_TOP	6
DELE	POP3_DELE	7
RSET	POP3_RSET	8
QUIT	POP3_QUIT	9
UIDL	POP3_UIDL	10
APOP	POP3_APOP	11
NOOP	POP3_NOOP	12
Server Reply	POP3_Reply	0

```
int POP3Message(CommInfo* machines, int POPCmd, int MessNum,
                int lines, MailInfo *Mailbox, int user)
```

Figure 4.8: POP3 Function Prototype

```
typedef struct MailboxInformation
 {
  char *ServerName;
  NetInfo ServerInfo;
  int NumOfUsers;
  char **UsrName;
  char **passwd;
  char **MD5;
  int *NumMess;
  char ***Messages;
  char ***UIDL;
  int **MessSize;
  NetInfo *Comps;
 } MailInfo;
```

Figure 4.9: MailInfo Data Type

4.6 POP3 Input

The application receives its input from a text file and fills in the MailInfo data structure for the application. This file contains all the necessary information that the application needs to run. It starts with the name and network information for the server. Next, it states the number of users that will be represented in the program. Then each user will have their user name, password and network information. After an individual user's information, the number of messages it has is stated. For each message, there will be the message size and the e-mail message itself, including the header information.

All of the information is not required for each implementation. If a researcher only wants the APOP authentication method, then the password data is not necessary and the password pointers will not be accessed. If the implementation does not require the UIDL command, then the UIDL character arrays are not necessary.

The e-mails used are actual e-mails which where copied into the input file. These messages may be sent a number of times, depending on how long the program runs. While this allows the user to easily generate input easily, it does make the traffic less authentic. It would be useful to develop an e-mail generator, yet that is out of the scope of this project.

4.7 Overall POP3 Handling

For any implementation, a general function should be created to control the traffic generation process. With the POP3 implementation, a general function controls the interaction between the clients and the server. First, the application takes the input from a given file which contains the user data to be used in the traffic. This file includes the login names, passwords, e-mail message information and actual e-mail messages. The function then pseudo-randomly initiates connections between the client and the server based on the amount of time passed. As shown in Figure 4.10, the function creates a TCP connection. Next the function logs the user onto the server with the login and password commands. Then, the list command is called and the user starts receiving and deleting the available e-mails using their respective commands. When no more e-mails are available, the function disconnects the TCP connection.

The data in the server is not modified during this process to allow for continuous traffic generation without having to generate new data. The network traffic will be generated for the duration of an input time from the user command line.

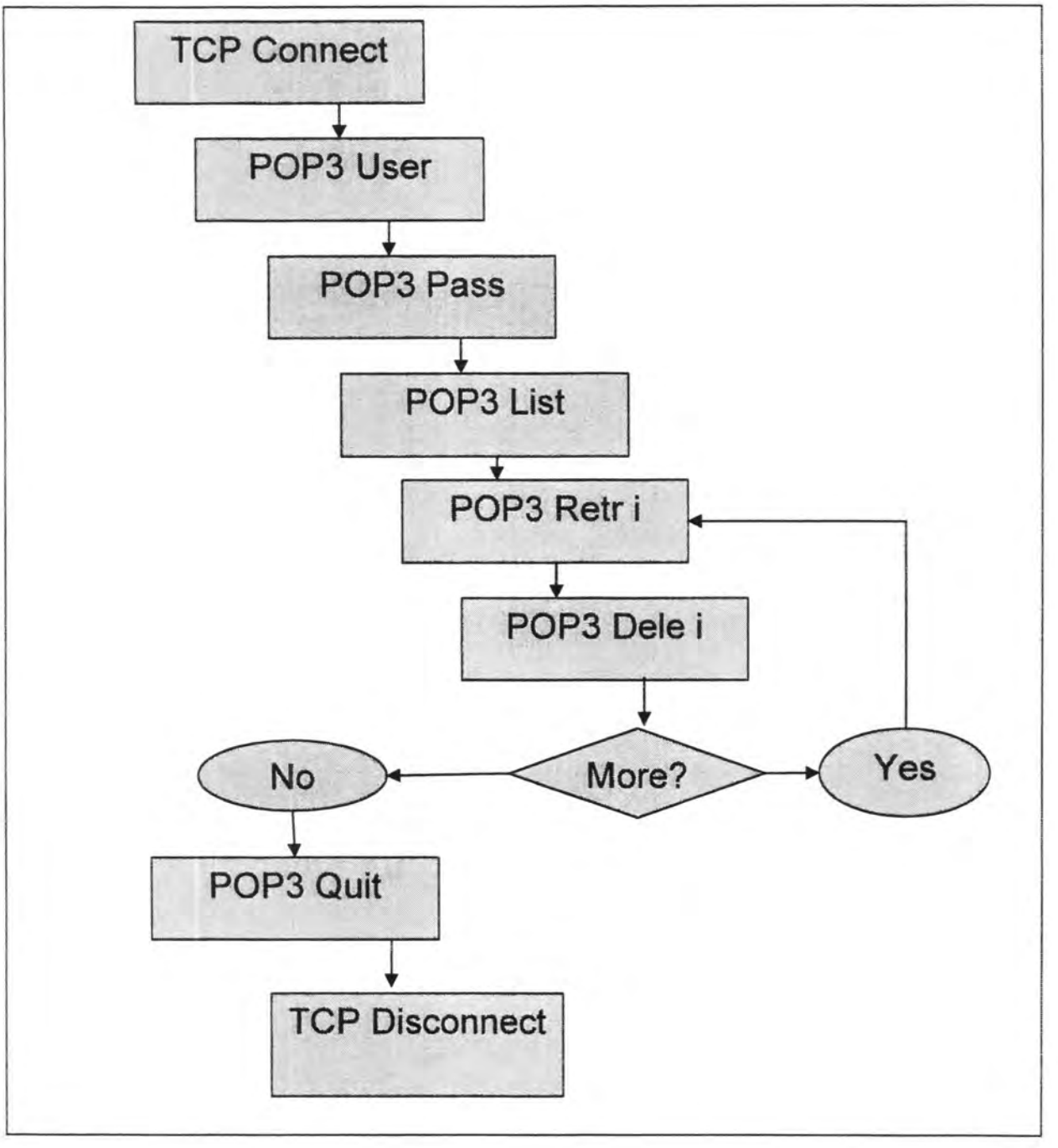

Figure 4.10: Sequence of function calls

CHAPTER 5. RESULTS

The Simple Network Imitator Program (SNIP) was developed to meet the requirements stated in the introduction and prove that it is possible to impersonate network communication. The code was developed on an x86 machine running Redhat Linux 9. Libnet version 1.1.2 was used to develop this proof-of-concept program. This chapter explains the results of the implementation.

5.1 Netcat Verification

The correctness of this program was primarily determined by having Ethereal sniffing SNIP traffic to determine if Ethereal recognizes the traffic as authentic POP3 communication. Baseline traffic was generated using Netcat to connect to a POP3 server. Netcat is a networking utility which connects to servers, opens communications then reads and writes data across network connections using the TCP/IP protocol [11]. Netcat has similar functionality to Telnet, but was chosen because Telnet sends the individual characters of the user's input across the network, quickly increasing the number of packets sent into the hundreds. On the other hand, Netcat does not send the user input across the network until is receives a carriage return. This reduces the number of packets for comparison greatly.

The traffic that was generated with Netcat was collected by Ethereal and is shown in Table 5.2. Ethereal filtered out all the traffic on the network except for the IP addresses of the machines running the baseline traffic generation. Each row in Table 5.2 represents one packet in the communication. Each row shows the packet number in the communication process, the source and destination IP addresses, the top protocol used (whether it be at the application layer or the transportation layer), and other packet information. The packet information fill first give packet errors, if there are any, and the information that the packet is caring. When the packet's top layer is the application layer, the information shows the data held by the application level packet. If the packet's top layer is the transportation layer, the information shows the purpose of the packet and the basic information, which is typically done when TCP acknowledgments are being sent.

Rows one through three of Table 5.1 shows the three way handshake between Netcat and the POP3 server, followed by the POP3 welcome information and request for user name.

The fifth packet is just a TCP acknowledgement to the POP3 welcome. Packets six through eleven are the USER and PASS commands with the appropriate server responses and client TCP acknowledgements. Packets 12 through 17 packets are is the LIST command followed by the appropriate responses. The RETR and QUIT command are called in packets 18 through 22 and 23 through the 29, respectively. Packets 25 through 28 are the four way disconnection, with packet 29 being the reset packet.

Table 5.2 shows the results of having Ethereal sniff traffic generated by SNIP. The sequence of packets is the same as the ones generated by the base traffic generation. The packets do have different sequence numbers and window numbers, since these are randomly generated numbers. The IP addresses for the SNIP generation where selected to be unrealistic address to show that they are controlled by the program. Ethereal does show that the packets are authentic TCP and POP3 traffic.

5.2 Netscape Mail Verification

Using Netcat gave the user complete control of the commands being sent, yet it is rare that a user would use this method to check their mail. Typically people use e-mail client programs, such as Netscape Mail, to easily handle their e-mail and control the communication process through the push of a few buttons. Since these programs are use significantly, an implementation of SNIP was created to produce equivalent noise traffic to that of Netscape Mail. The output from Ethereal's sniffing Netscape's traffic is shown in Table 5.3. Table 5.4 shows Ethereal's output from sniffing the SNIP representation of Netscape Mail receiving an e-mail. It is apparent that these two traffic generators are equivalent.

The Netscape Mail implementation of SNIP differed from the Netcat implementation in a couple of ways. First, the commands to retrieve the data were different. Netscape Mail used the UILD and STAT commands, while Netcat didn't. Also, Netscape Mail did not have to send specific TCP acknowledgement packets like Netcat did. These differences were handled in the top function which controls the overall handling of the different protocol functions.

Table 5.1: Netcat user generated traffic

No.	Source	Destination	Protocol	Info
1	129.186.152.249	129.186.215.40	TCP	36646 > pop3 [SYN] Seq=3443380522 Ack=0 Win=5840 Len=0
2	129.186.215.40	129.186.152.249	TCP	pop3 > 36646 [SYN, ACK] Seq=4018811733 Ack=3443380523 Win=57344 Len=0
3	129.186.152.249	129.186.215.40	TCP	36646 > pop3 [ACK] Seq=3443380523 Ack=4018811734 Win=5840 Len=0
4	129.186.215.40	129.186.152.249	POP	Response: +OK QPOP (version 2.53) at spock.ee.iastate.edu starting. <9584.1080844989@spock.ee.iastate.edu>
5	129.186.152.249	129.186.215.40	TCP	36646 > pop3 [ACK] Seq=3443380523 Ack=4018811833 Win=5840 Len=0
6	129.186.152.249	129.186.215.40	POP	Request: user brettm
7	129.186.215.40	129.186.152.249	POP	Response: +OK Password required for brettm.
8	129.186.152.249	129.186.215.40	TCP	36646 > pop3 [ACK] Seq=3443380535 Ack=4018811868 Win=5840 Len=0
9	129.186.152.249	129.186.215.40	POP	Request: pass test01
10	129.186.215.40	129.186.152.249	POP	Response: +OK brettm has 3 messages (3158 octets).
11	129.186.152.249	129.186.215.40	TCP	36646 > pop3 [ACK] Seq=3443380547 Ack=4018811910 Win=5840 Len=0
12	129.186.152.249	129.186.215.40	POP	Request: list
13	129.186.215.40	129.186.152.249	POP	Response: +OK 3 messages (3158 octets)
14	129.186.152.249	129.186.215.40	TCP	36646 > pop3 [ACK] Seq=3443380552 Ack=4018811940 Win=5840 Len=0
15	129.186.215.40	129.186.152.249	POP	Continuation
16	129.186.152.249	129.186.215.40	TCP	36646 > pop3 [ACK] Seq=3443380552 Ack=4018811967 Win=5840 Len=0
17	129.186.215.40	129.186.152.87	TCP	telnet > 3145 [ACK] Seq=2201419551 Ack=3585168974 Win=58400 Len=0
18	129.186.152.249	129.186.215.40	POP	Request: retr 1
19	129.186.215.40	129.186.152.249	POP	Response: +OK 1104 octets
20	129.186.152.249	129.186.215.40	TCP	36646 > pop3 [ACK] Seq=3443380559 Ack=4018811984 Win=5840 Len=0
21	129.186.215.40	129.186.152.249	POP	Continuation
22	129.186.152.249	129.186.215.40	TCP	36646 > pop3 [ACK] Seq=3443380559 Ack=4018813091 Win=7749 Len=0
23	129.186.152.249	129.186.215.40	POP	Request: quit
24	129.186.215.40	129.186.152.249	POP	Response: +OK Pop server at spock.ee.iastate.edu signing off.
25	129.186.152.249	129.186.215.40	TCP	36646 > pop3 [ACK] Seq=3443380564 Ack=4018813144 Win=7749 Len=0
26	129.186.215.40	129.186.152.249	TCP	pop3 > 36646 [FIN, ACK] Seq=4018813144 Ack=3443380564 Win=57920 Len=0
27	129.186.152.249	129.186.215.40	TCP	36646 > pop3 [FIN, ACK] Seq=3443380564 Ack=4018813145 Win=7749 Len=0
28	129.186.215.40	129.186.152.249	TCP	pop3 > 36646 [ACK] Seq=4018813145 Ack=3443380565 Win=57920 Len=0

Table 5.2: SNIP generated traffic for Netcat

No.	Source	Destination	Protocol	Info
1	1.2.3.4	4.3.2.1	TCP	34 > pop3 [SYN] Seq=1247443331 Ack=0 Win=32767 Len=0
2	4.3.2.1	1.2.3.4	TCP	pop3 > 34 [SYN, ACK] Seq=2015967873 Ack=1247443332 Win=32767 Len=0
3	1.2.3.4	4.3.2.1	TCP	34 > pop3 [ACK] Seq=1247443332 Ack=2015967874 Win=32767 Len=0
4	4.3.2.1	1.2.3.4	POP	Response: +OK QPOP (version 2.53) at spock.ee.iastate.edu starting. <8379.1080829469@spock.ee.iastate.edu>
5	1.2.3.4	4.3.2.1	TCP	34 > pop3 [ACK] Seq=1247443333 Ack=2015967971 Win=32767 Len=0
6	1.2.3.4	4.3.2.1	POP	Request: user brettm
7	4.3.2.1	1.2.3.4	POP	Response: +OK Password required for brettm.
8	1.2.3.4	4.3.2.1	TCP	34 > pop3 [ACK] Seq=1247443344 Ack=2015968004 Win=32767 Len=0
9	1.2.3.4	4.3.2.1	POP	Request: pass test01
10	4.3.2.1	1.2.3.4	POP	Response: +OK brettm has 3 messages (3158 octets).
11	1.2.3.4	4.3.2.1	TCP	34 > pop3 [ACK] Seq=1247443355 Ack=2015968044 Win=32767 Len=0
12	1.2.3.4	4.3.2.1	POP	Request: list
13	4.3.2.1	1.2.3.4	POP	Response: +OK brettm has 3 messages (3158 octets).
14	1.2.3.4	4.3.2.1	TCP	34 > pop3 [ACK] Seq=1247443359 Ack=2015968084 Win=32767 Len=0
15	4.3.2.1	1.2.3.4	POP	Continuation
16	1.2.3.4	4.3.2.1	TCP	34 > pop3 [ACK] Seq=1247443359 Ack=2015968106 Win=32767 Len=0
17	1.2.3.4	4.3.2.1	POP	Request: retr 1
18	4.3.2.1	1.2.3.4	POP	Response: +OK 1104 octet.
19	1.2.3.4	4.3.2.1	TCP	34 > pop3 [ACK] Seq=1247443365 Ack=2015968121 Win=32767 Len=0
20	4.3.2.1	1.2.3.4	POP	Continuation
21	1.2.3.4	4.3.2.1	TCP	34 > pop3 [ACK] Seq=1247443365 Ack=2015969206 Win=32767 Len=0
22	1.2.3.4	4.3.2.1	POP	Request: quit
23	4.3.2.1	1.2.3.4	POP	Response: +OK Pop server at spock.ee.iastate.edu signing off.
24	1.2.3.4	4.3.2.1	TCP	34 > pop3 [ACK] Seq=1247443369 Ack=2015969257 Win=32767 Len=0
25	4.3.2.1	1.2.3.4	TCP	pop3 > 34 [FIN, ACK] Seq=2015969257 Ack=1247443369 Win=32767 Len=0
26	1.2.3.4	4.3.2.1	TCP	34 > pop3 [ACK] Seq=1247443369 Ack=2015969258 Win=32767 Len=0
27	1.2.3.4	4.3.2.1	TCP	34 > pop3 [FIN, ACK] Seq=1247443369 Ack=2015969258 Win=32767 Len=0
28	4.3.2.1	1.2.3.4	TCP	pop3 > 34 [ACK] Seq=2015969258 Ack=1247443370 Win=32767 Len=0

Table 5.3: Netscape mail generated traffic

No.	Source	Destination	Protocol	Info
1	129.186.152.56	129.186.215.40	TCP	1072 > pop3 [SYN] Seq=3749790475 Ack=0 Win=64240 Len=0
2	129.186.215.40	129.186.152.56	TCP	pop3 > 1072 [SYN, ACK] Seq=3487214969 Ack=3749790476 Win=57344 Len=0
3	129.186.152.56	129.186.215.40	TCP	1072 > pop3 [ACK] Seq=3749790476 Ack=3487214970 Win=64240 Len=0
4	129.186.215.40	129.186.152.56	POP	Response: +OK QPOP (version 2.53) at spock.ee.iastate.edu starting. <12399.1086695818@spock.ee.iastate.edu>
5	129.186.152.56	129.186.215.40	POP	Request: USER brettm
6	129.186.215.40	129.186.152.56	POP	Response: +OK Password required for brettm.
7	129.186.152.56	129.186.215.40	POP	Request: PASS test01
8	129.186.215.40	129.186.152.56	POP	Response: +OK brettm has 1 message (769 octets).
9	129.186.152.56	129.186.215.40	POP	Request: STAT
10	129.186.215.40	129.186.152.56	POP	Response: +OK 1 769
11	129.186.152.56	129.186.215.40	POP	Request: LIST
12	129.186.215.40	129.186.152.56	POP	Response: +OK 1 messages (769 octets)
13	129.186.152.56	129.186.215.40	TCP	1072 > pop3 [ACK] Seq=3749790514 Ack=3487215185 Win=64025 Len=0
14	129.186.215.40	129.186.152.56	POP	Continuation
15	129.186.152.56	129.186.215.40	POP	Request: UIDL
16	129.186.215.40	129.186.152.56	POP	Response: +OK uidl command accepted.
17	129.186.152.56	129.186.215.40	TCP	1072 > pop3 [ACK] Seq=3749790520 Ack=3487215223 Win=63987 Len=0
18	129.186.215.40	129.186.152.56	POP	Continuation
19	129.186.152.56	129.186.215.40	POP	Request: RETR 1
20	129.186.215.40	129.186.152.56	POP	Response: +OK 769 octets
21	129.186.152.56	129.186.215.40	TCP	1072 > pop3 [ACK] Seq=3749790528 Ack=3487215278 Win=63932 Len=0
22	129.186.215.40	129.186.152.56	POP	Continuation
23	129.186.152.56	129.186.215.40	POP	Request: DELE 1
24	129.186.215.40	129.186.152.56	POP	Response: +OK Message 1 has been deleted.
25	129.186.152.56	129.186.215.40	POP	Request: QUIT
26	129.186.215.40	129.186.152.56	POP	Response: +OK Pop server at spock.ee.iastate.edu signing off.
27	129.186.215.40	129.186.152.56	TCP	pop3 > 1072 [FIN, ACK] Seq=3487216136 Ack=3749790542 Win=58400 Len=0
28	129.186.152.56	129.186.215.40	TCP	1072 > pop3 [ACK] Seq=3749790542 Ack=3487216137 Win=63074 Len=0
29	129.186.152.56	129.186.215.40	TCP	1072 > pop3 [FIN, ACK] Seq=3749790542 Ack=3487216137 Win=63074 Len=0
30	129.186.215.40	129.186.152.56	TCP	pop3 > 1072 [ACK] Seq=3487216137 Ack=3749790543 Win=58400 Len=0

Table 5.4: SNIP generated traffic for Netscape

No.	Source	Destination	Protocol	Info
1	1.2.3.4	4.3.2.1	TCP	34 > pop3 [SYN] Seq=1295804997 Ack=0 Win=32767 Len=0
2	4.3.2.1	1.2.3.4	TCP	pop3 > 34 [SYN, ACK] Seq=1809603373 Ack=1295804998 Win=32767 Len=0
3	1.2.3.4	4.3.2.1	TCP	34 > pop3 [ACK] Seq=1295804998 Ack=1809603374 Win=32767 Len=0
4	4.3.2.1	1.2.3.4	POP	Response: +OK QPOP (version 2.53) at spock.ee.iastate.edu starting. <8379.1080829469@spock.ee.iastate.edu>
5	1.2.3.4	4.3.2.1	POP	Request: USER brettm
6	4.3.2.1	1.2.3.4	POP	Response: +OK Password required for brettm.
7	1.2.3.4	4.3.2.1	POP	Request: PASS test01
8	4.3.2.1	1.2.3.4	POP	Response: +OK brettm has 1 messages (1009 octests).
9	1.2.3.4	4.3.2.1	POP	Request: STAT
10	4.3.2.1	1.2.3.4	POP	Response: +OK 1 1009
11	1.2.3.4	4.3.2.1	POP	Request: LIST
12	4.3.2.1	1.2.3.4	POP	Response: +OK brettm has 1 message (1009 octests)
13	1.2.3.4	4.3.2.1	TCP	34 > pop3 [ACK] Seq=1295805029 Ack=1809603594 Win=32767 Len=0
14	4.3.2.1	1.2.3.4	POP	Continuation
15	1.2.3.4	4.3.2.1	POP	Request: UIDL
16	4.3.2.1	1.2.3.4	POP	Response: +OK uidl command accpeted.
17	1.2.3.4	4.3.2.1	TCP	34 > pop3 [ACK] Seq=1295805033 Ack=1809603627 Win=32767 Len=0
18	4.3.2.1	1.2.3.4	POP	Continuation
19	1.2.3.4	4.3.2.1	POP	Request: retr 1
20	4.3.2.1	1.2.3.4	POP	Response: +OK 1009 octet
21	1.2.3.4	4.3.2.1	TCP	34 > pop3 [ACK] Seq=1295805039 Ack=1809603641 Win=32767 Len=0
22	4.3.2.1	1.2.3.4	POP	Continuation
23	1.2.3.4	4.3.2.1	POP	Request: dele 1
24	4.3.2.1	1.2.3.4	POP	Response: +OK Message 1 has been deleted.
25	1.2.3.4	4.3.2.1	POP	Request: quit
26	4.3.2.1	1.2.3.4	POP	Response: +OK Pop server at spock.ee.iastate.edu signing off.
27	4.3.2.1	1.2.3.4	TCP	pop3 > 34 [FIN, ACK] Seq=1809604676 Ack=1295805080 Win=32767 Len=0
28	1.2.3.4	4.3.2.1	TCP	34 > pop3 [ACK] Seq=1295805080 Ack=1809604677 Win=32767 Len=0
29	1.2.3.4	4.3.2.1	TCP	34 > pop3 [FIN, ACK] Seq=1295805080 Ack=1809604677 Win=32767 Len=0
30	4.3.2.1	1.2.3.4	TCP	pop3 > 34 [ACK] Seq=1809604677 Ack=1295805081 Win=32767 Len=0

CHAPTER 6. SUMMARY

SNIP was designed and developed in response to the need for test bed networks to behave more like production networks. The end product proved that traffic from both the client and the server can be imitated by a single machine without making the machine vulnerable to attacks as a result of the tool. The rest of this chapter explains how the tool meets the requirements given and then details the future of SNIP.

6.1 Compliance with Objectives

The objectives of this tool were to prove that an application can create network traffic identical to traffic seen on a production network. The traffic will only affect the immediate network it is being produced on and will not bridge a second layer interconnect, such as a router. The application should also not create another security threat to the machine it is running on. Finally, the code for this application should be component based so it can be easily modified and reused if needed. As explained below, all of these objectives have been met by the tool that has been created.

The main objective of this tool was to prove that network traffic can be imitated by a single source. SNIP has shown that it is possible by the results given. The network traffic produced by this tool is viewed as typical network traffic.

The traffic generated by this tool will not cross a second layer network interconnection because the MAC Addresses are spoofed. Since no machine on the network has the MAC Address being used in the packet, after the packet is created and sent, the only machines that will pick up the traffic are machines that are promiscuously sniffing the traffic. This means routers and switches will not pick up the packet and reproduce it on another network. There is a possibility that a machine on the network could have one of the spoofed addresses, but there are nearly 2^{48} different addresses possibilities, so it is highly unlikely that this will occur.

The next objective was that the application does not create additional security threats to the machines they are running on. This is helpful when the test bed networks are being used for penetration tests. The application should not provide another vulnerability that can be used to access restricted machines. Because this application does not process input from

other machines, it does not create a security threat. However, it is possible that the traffic produced by this application can add to a DOS attack flooding the network and making it difficult for other machines to use the network.

Finally, the code is based off of multiple functions making it easier to reuse the code. These functions have been placed in a separate source file with a corresponding header file. Also, the Libnet Library that is used extensively within the code is supported on multiple platforms, such as Linux and Microsoft Windows XP.

6.2 ISEAGE

This tool has the ability to be run as one of the custom tools on ISEAGE, as shown in the system architecture layout in Figure 6.1. This test bed network will be creating a virtual Internet and there will be several subnets within this virtual Internet. The traffic being generated by these subnets can be produces by this tool.

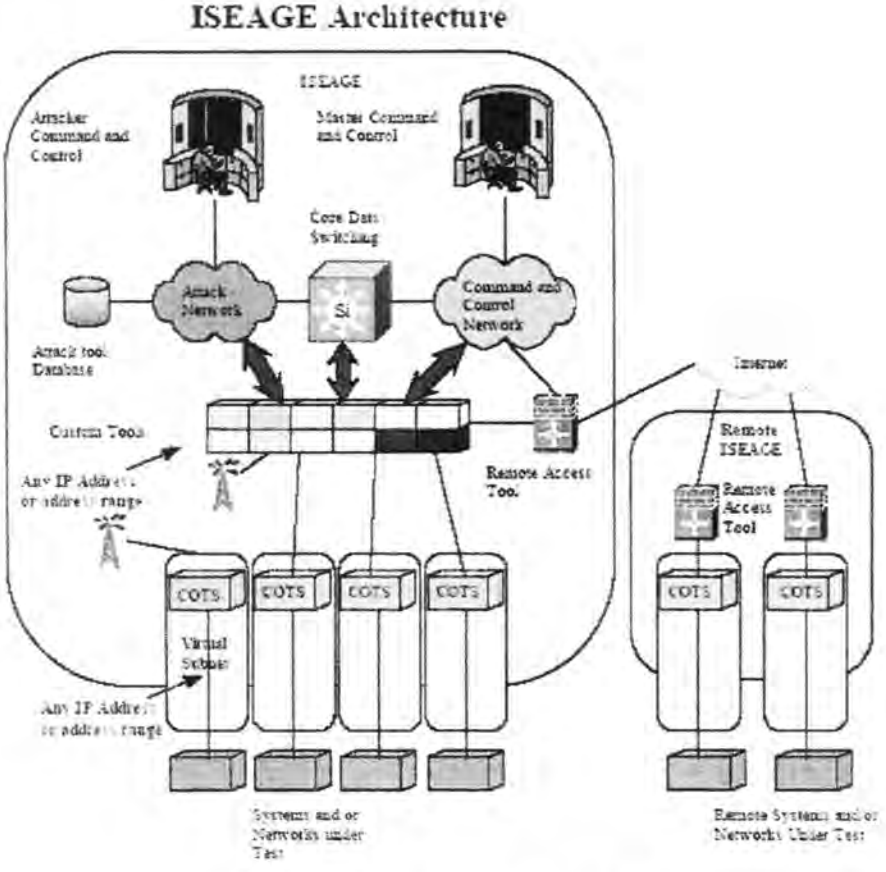

Figure 6.1: ISEAGE Architecture Layout

6.3 Future Work

This project has just begun and is still in its infancy. Here are several ways to improve the usefulness of this tool.

6.3.1 Other Application Protocols

While the tool has been developed to produce POP3 traffic, there are several more application layer protocols that can be implemented to improve the effectiveness of this tool. Other possible protocols could be the Hyper-text transfer protocol (HTTP), the file transfer protocol (FTP), the simple mail transfer protocol (SMTP), the simple network management protocol (SNMP), or the internet control message protocol (ICMP) to name a few. Also, a larger variety of application layer protocols may be helpful to researchers. Each additional protocol will not only require its own protocol implementation functions, but will need its own control unit as well.

6.3.2 Server Specific Implementations

While all the protocols that are used are the same, each implementation of these protocols may vary slightly. Typical users might not see the difference between Microsoft's web server or the Apache web server, their packets may vary just slightly even if they are hosting the same webpage. If someone was trying to produce traffic to represent a specific server, they would have to adjust the tool producing the traffic to match the specifics of that particular server. This would be helpful if the implementer of the tool was either testing an application identification tool or running a honeynet attempting to get someone to believe that a particular application is running.

6.3.3 Remote Administration

One of the planned uses for the tool is to run on ISE AGE, which will have several computers possibly running this application. Because of this, it of interest to add remote

administration to this application so the user does not have to have contact with the machine. This addition would also allow people to monitor their test bed networks while in their office.

6.3.4 Graphical User Interface

A graphical user interface (GUI) could make this tool more user friendly, allowing it to be used by more people. A GUI could make it easier to format and change the user input and manage the systems. It could also be coordinated with control protocol, making remote administration easier to handle.

6.4 Conclusion

SNIP was successful in proving that network traffic can be imitated from a single source. It can be effectively used to improve test bed networks by generating realistic network traffic. While this implementation is just a proof of concept, it provides a good foundation for the future development of this tool.

ACKNOWLEDGEMENTS

I would like to thanks Dr. Doug Jacobson for all the help and opportunities he has given me over the past five years. I would also like to thanks Dr. Jim Davis and Paul Heaberlin for their assistance in writing my thesis.

I would like to thank my parents, Kent and Marilee, and my brother Ryan, for all the love and support they have given me over the years. They have put up with a lot from me, and I wouldn't be able to complete what I have done so far in life without them.

Finally, I would like to thank my friends who have put up with me during my college experience. You've helped me enjoy the experience.

REFERENCES

[1] Academic Centers of Excellence, Date accessed 3/20/2004, www.nsa.gov/ia

[2] CERT Statistics, Date accessed 4/13/2004, www.cert.org

[3] Ethereal, Date accessed 4/13/2004 www.ethereal.com

[4] Forouzan, Behrouz A. TCP/IP Protocol Suit. McGraw Hill, New York, NY, 2003

[5] Information Assurance, Date accessed 4/13/2004, www.iac.iastate.edu

[6] ISEAGE, Date accessed 4/13/2004, www.iseage.issl.org

[7] Kendall, Kristopher "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems", S.M. Thesis, Department of Electrical Engineering and Computer Science, MIT, June 1999

[8] Korba, Jonathan "Windows NT Attacks for the Evaluation of Intrusion Detection Systems", S.M. Thesis, Department of Electrical Engineering and Computer Science, MIT, June 2000

[9] Libes, D. Exploring Expect: A Tcl-based Toolkit for Automating Interactive Programs, O'Reilly & Assoc., Sebastapol, CA, 1998

[10] Myers, J. and M. Rose "Post Office Protocol – Version 3 (RFC1939)", Date accessed 6/8/2004, www.faqs.org/rfcs/rfc1939.html

[11] Netcat, Date accessed 4/13/2004, netcat.sourceforge.net/

[12] Puketza, Nicholas, Mandy Chung, Ronald Olsson, and Biswanath Mukherjee. "A Software Platform for Testing Intrusion Detection Systems." Technical report, University of California, Davis, Department of Computer Science, Davis, CA, September 1995.

[13] Rossey, Lee M., Robert K. Cunningham, David J. Fried, Jesse C. Rabek, Richard P. Lippmann, Joshua W. Haines, and Marc A. Zissman. "LARIAT: Lincoln Adaptable Real-time Information Assurance Testbed", Aerospace Conference Proceedings, 2002. IEEE, Vol.6, Iss., 2002 Pages: 6-2671-2676, 6-2678- 6-2682 vol.6

[14] Schiffman, Mike D. Building Open Source Network Security Tools: Components and Techniques. Wiley Publishing, Inc. Indianapolis, IN, 2003

[15] Schiffman, Mike D., Libnet, Date accessed 4/13/2004, www.packetfactory.com/libnet