

**Ministry of Higher Education
and Scientific Research
Al-Nahrain University
College of Science
Department of Computer Science**



Hiding Text in Sequence Number Field of TCP/IP

A Thesis

Submitted to the College of Science / Al-Nahrain University in partial
fulfillment of the requirement for the degree of Master of Science in
Computer science

By

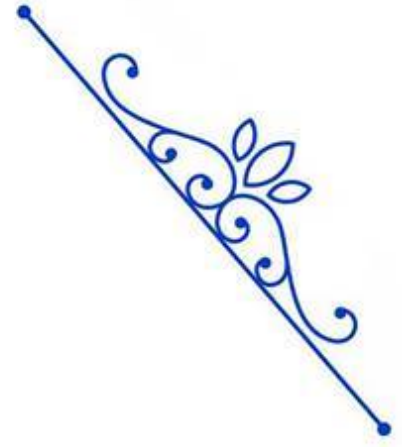
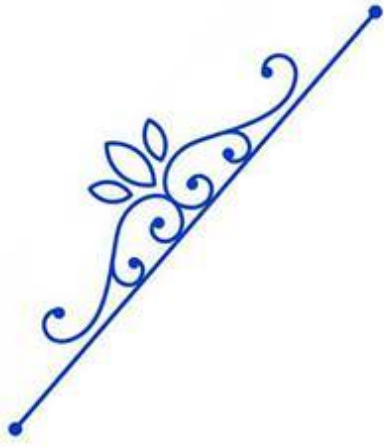
Abeer Eesa Abed

(B.Sc Computer Science / College of Science / Al- Nahrain University, 2009)

**Supervised by
Dr. Jamal M. Kadhim**

January 2017

Rabie Al- Akhar 1438



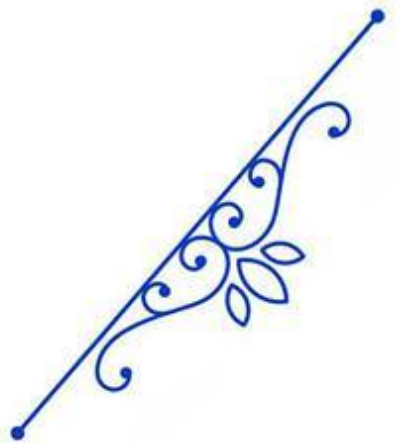
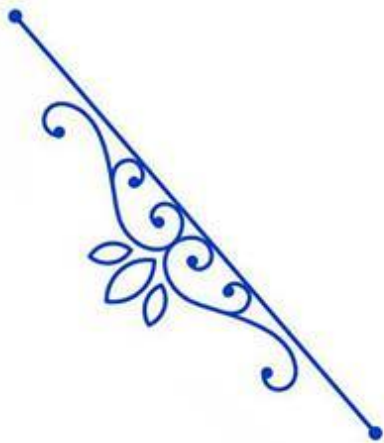
بِسْمِ اللّٰهِ الرَّحْمٰنِ الرَّحِیْمِ

« قَالُوا سُبْحَانَكَ لَا عِلْمَ لَنَا إِلَّا مَا عَلَّمْتَنَا
إِنَّكَ أَنْتَ الْعَلِيمُ الْحَكِيمُ »

صدق الله العظيم

سورة البقرة

الآية (٣٢)



Supervisor Certification

I certify that this thesis entitled "**Hiding Text in Sequence Number Field of TCP/IP**" was prepared by "**Abeer Eesa Abed**" under my supervision at the College of Science /Al-Nahrain University as a partial fulfillment of the requirements for the Degree of Master of Science in computer science

Signature:



Name : **Dr. Jamal Muhammad Kadhim**

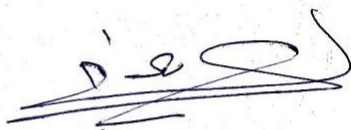
Scientific Degree: **Lecturer**

Address: College of Science /Al-Nahrain University

Date : **24/7/2017**

In view of the available recommendations, I forward this **thesis** for debate by the examination committee.

Signature:



Name : **Dr. Ban Nadeem Dhannoon**


Scientific Degree: **Professor**

Address: Head of Computer Science Department / Al-Nahrain University

Date : **27/8/2017**

Committee Certification

We, the examining committee certify that we have read this thesis entitled "Hiding Text in Sequence Number Field of TCP/IP" and examined the student "Abeer Eesa Abed" in its content and that in our opinion; it is accepted for the Degree of Master of Science in Computer Science.

Signature: 


Name: **Dr. Ban Nadeem Dhannoon**

Scientific Degree: **Professor**

Address: College of Science /Al-Nahrain University

Date: 27/8/2017

(Chairman)

Signature: 


Name: **Dr. Abeer T. Malood**

Scientific Degree: **Assistant Prof.**

Address: Dept. of Computer Science
University of Technology

Date: 24/8/2017

(Member)

Signature: 

Name: **Dr. Raghad A. Azeez**

Scientific Degree: **Assistant Prof.**

Address: College of Education
for Human Science – Ibn
Rushd / Baghdad University

Date: 23/7/2017

(Member)

Signature: 

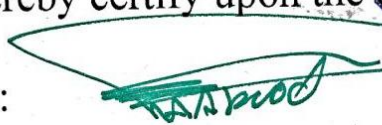
Name: **Dr. Jamal Muhammad Kadhim**

Scientific Degree: **Lecturer**

Address: College of Science /Al-Nahrain University

Date: 24/7/2017

I, hereby certify upon the decision of the examining committee

Signature: 

Name: **Dr. Hadi M. A. Abood**

Scientific Degree: **Professor**

Address: Dean of College of Science /Al-Nahrain University

Date: 19/9/2017



Dedicated To

Allah...

*Who was with me each pulse of
my heart,*

My Family, and

Every One Prayer for me ...

Abeer

Acknowledgement

*I would like to thank Allah, who set me up stars that glittered in the sky of my studies starter with my supervisor **Dr. Jamal Mohummed Kadhim**. Deep appreciation and many thanks to him for suggestion the subject, guidance, valuable assistance and advice,*

*Grateful thanks to the Head of Computer Science Department **Dr. Ban Nadeem** for kindness and attention,*

*Deep gratitude and honest thanks to **Dr. Haithem Abdeulateef** and **Dr. Taha S. Bashagha** for their encouraging words and attention,*

*Thanks a lot to **staff of computer science department** and **my friends** for attention,*

*And ended with **my family** specially **my father** and **my mother**. I owe deeply grateful to them for their patience and help during my studies and every one helped me and prayer for me.*

Abeer

ABSTRACT

The exchanging process inside Local Area Network (LAN) or through Internet may be exposed to be stolen, altered or damaged by baleful person who was represented as real threats to transport process and also to information especially if this information was sensitive, important and must be accessed by only authorized person. Wherefore this data must be secured against such threats. Many ideas was suggested under security concept for protecting data from this threats such as hiding content of the message sent which was named cryptography or concealment the existence of such message which was named steganography.

Two ways were suggested to hide this data. One of them used the source port and destination port fields of the Transmission Control Protocol (TCP) header as the Stego key. And the other use the combination of source and source port fields of the Transmission Control Protocol (TCP) header with the protocol and version fields of the Internet Protocol (IP) header. The process is summarized by the implementation of the exclusive OR (XOR) between those data required to be hidden with the STEGO key. A sequence number field was selected from the Transmitter Control Protocol (TCP) to be the carrier for hidden data. Four characters are included in this field and sent in one connection.

The suggested methods differ from the existing methods. One of them was sent one character through one connection while in the proposed methods four characters are sent. In addition to this difference, the stego key that was used also differed. Because the constant value will be collected with the ASCII character code. While in the proposed methods are variable and the collection process is not used but XOR is used. Although in the other methods four characters have been sent, but they used many resources for execution because the characters have been compressed and encrypted.

List of Abbreviations

<u>Abbreviation</u>	<u>Meaning</u>
ACK	Acknowledgement
AES	Advanced Encryption Standard
API	Application Programming Interface
ARPANET	Advanced Research Projects Agency Network
ATM	Asynchronous Transfer Mode
DES	Data Encryption Standards
DF	Do not Fragment
DNS	Domain Name System
DoD	Department of Defense
DS	Differentiated Services
FIN	Finish
FTP	File Transfer Protocol
FSM	Finite State Machine
HICCUPS	Hidden Communication System for Corrupted Networks
HTTP	Hyper Text Transfer Protocol
ICMP	Internet Control Message Protocol
IGMP	Internet Group Management Protocol
IHL	IPHeaderLength
IP	Internet Protocol
IPv4	IP version 4

ISO	International Standards Organization
ISN	Initial Sequence Number
MF	More Fragment
MTU	Maximum Transfer Unit
MSS	Maximum Segment Size
NFS	Network File System
NIC	Network Interface Card
NS	Network systems
OSI	Open Systems Interconnect
OSPF	Open Shortest Path First
PPP	Point-to-Point Protocol
PSH	Push
RSA	Rivest-Shamir-Adleman
RST	Reset
SLIP	Serial Line Internet Protocol
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SYN	Synchronize
SYN-ACK	Synchronize- Acknowledgement
TCP	Transmission Control Protocol
ToS	Type of Service
UDP	User Datagram Protocol
URG	Urgent
VER	Version

Table of Algorithms

<u>Algorithm Name</u>		<u>Page number</u>
Algorithm (3.1)	Generate Packet	34
Algorithm (3.2)	CreateIPHeader	35
Algorithm (3.3)	Create TCPheader	36
Algorithm (3.4)	Generate sequence number with srcdstports	38
Algorithm (3.5)	Generate sequence number with srcport, destport, protocol, and version	40
Algorithm (3.6)	Receive SYN -ACK packet	42
Algorithm (3.7)	Packet Capturing and Analyzing at Layer 3	43
Algorithm (3.8)	Capturing Packet Function	44
Algorithm (3.9)	Capturing and Analyzing at layer 2	46
Algorithm (3.10)	Data Extraction using acknowledgement number with srcdstports	49
Algorithm (3.11)	Data Extraction with srcport, destport, protocol and version	50

Table of Figures

<u>Figure Name</u>		<u>Page Number</u>
Figure (2.1)	TCP / IP Layers and Protocols	8
Figure (2.2)	Encapsulation and Decapsulation	9
Figure (2.3)	IP header format	11
Figure (2.4)	TCP header format	14
Figure (2.5)	Socket	17
Figure (2.6)	Types of Sockets	19
Figure (2.7)	Client – Server connection through TCP socket	20
Figure (2.8)	Three way handshake	22
Figure (2.9)	nc command	24
Figure (2.10)	watch with netstat commands	25
Figure (2.11)	nc with watch and netstat	25
Figure (2.12)	watch with netstat commands after executing	25
Figure (3.1)	the proposed system model	33
Figure (3.2)	Stego-key from TCP Header	37
Figure (3.3)	Stego-key from IP and TCP Headers	39
Figure (3.4)	Data Extraction Module	48
Figure (4.1)	source and destination addresses with the data- First method	53

Figure (4.2)	source and destination ports – First method	53
Figure (4.3)	data hiding process – First method – First execution	53
Figure (4.4)	Capturing and Analyzing – Layer 3- First execution	54
Figure (4.5)	Data extraction process – First method- First execution	54
Figure (4.6)	source and destination ports – First method – Second execution	55
Figure (4.7)	Data hiding process – First method – Second execution	55
Figure (4.8)	Capturing and Analyzing – Ethernet – First method- Second execution	55
Figure (4.9)	Data extraction process – First method- Second execution	56
Figure (4.10)	source and destination addresses, version, protocol, data- Second method	56
Figure (4.11)	source and destination ports- Second method- First execution	56
Figure (4.12)	Data hiding process – Second method – First execution	57
Figure (4.13)	Capturing and Analyzing – Layer 3- Second method – First execution	57
Figure (4.14)	Data extraction process – Second Method- First execution	57
Figure (4.15)	source and destination ports- Second method-	58

	Second execution	
Figure (4.16)	Data hiding process – Second method – Second execution	58
Figure (4.17)	Capturing and Analyzing – Ethernet – Second method- Second execution	59
Figure (4.18)	Data extraction process – Second Method- Second execution	59
Figure (4.19)	Changing iptables values	60
Figure (4.20)	SYN packet	60
Figure (4.21)	IPHeader of SYN packet	61
Figure (4.22)	TCPHeader of SYN packet	62
Figure (4.23)	SYN-ACK packet	63
Figure (4.24)	IPHeader of SYN-ACK packet	64
Figure (4.25)	TCPHeader of SYN-ACK packet	64
Figure (4.26)	Execution of program after changing iptables	65
Figure (4.27)	Execution of program without changing iptables	66
Figure (4.28)	TCPHeader of RST packet	67

Table of Contents

Chapter one

General Introduction

1.1 Introduction	1
1.2 Literature Review	2
1.3 Aim of Thesis	3
1.4 Thesis Layout	3

Chapter Two

TCP / IP and Steganography under Linux Environment

2.1 Introduction	5
2.2 TCP / IP Reference Model	5
2.2.1 Internet Protocol (IP)	10
2.2.2 Transmission Control Protocol	13
2.2.3 Packet Analysis	16
2.3 Client- Server Architecture	16
2.3.1 Application Programming Interface	17
2.3.2 Server	23
2.4 Information Hiding	25
2.4.1 History of Steganography	26
2.4.2 Modern Steganography	26
2.5 Linux	28

Chapter Three

Steganography in TCP/IP System Design and Implementation

3.1 Introduction	30
3.2 Text Hiding Using Sequence Number field Implementation	30
3.3 System Model	32
3.4 Sender Model	34
3.4.1 Generate Packet	34
3.4.2 Send SYN packet	41
3.4.3 Receive SYN-ACK packet	41
3.5 Receiver Model	42
3.5.1 Capturing and Analyzing process	43
3.5.2 Data Extraction	47
3.6 Server Model	51
Chapter Four	
Results	
4.1 Introduction	52
4.2 Results	52
Chapter Five	
Discussion, Conclusions, and Future Work	
5.1 Discussion	68
5.2 Conclusions	69
5.3 Future Work	70
References	71

CHAPTER ONE

CHAPTER ONE GENERAL INTRODUCTION

1.1 Introduction

After computer appearance and pervasion in most institutes like companies, universities, and homes, the need for network appeared. Since these places had limited hardware resources like printers, scanners...etc, so individuals, especially employees needed to share these resources because each employee had no need to have private printer. Also, after growth of most institutes which was represented by having different branches distributed in different places like companies and universities, so the individuals really need to communicate and at the same time to exchange important information to do their work, so the need to connect these institutes by network became necessary[TAN03].

Network in simple case connect at least two computers using connection means Network may be built in homogeneous form (i.e. every computer in network has the same operating system ,Network Interface Card (NIC) ... etc like LAN or in heterogeneous form (i.e. every computer has different operating system, NIC, ... etc) like Internet . As a result, this development in communication means leads to exchange huge information.

The exchanging process inside LAN or through Internet may be exposed to be stolen, altered or damaged by baleful person who was represented as real threats to transport process and also to information especially if this information was sensitive, important and must be accessed by only authorized person.

Wherefore this data must be secured against such threats. Many ideas was suggested under security concept for protecting data from this threats such as hiding

content of the message sent which was named cryptography or concealment the existence of such message which was named steganography.

1.2 Literature Review

Previous works show many techniques used to hide data in TCP/IP protocol suite either by using reserved or unused bits in headers and payload or by using synchronization time of the packets or combine between them.

Rowland [ROW97] applied his ideas to hide data using Initial *Sequence Number Field (ISN)* (16-bit) of TCP by *multiplying* ASCII of each character with $(65536 * 256)$ to generate number which was placed as a sequence number value for each connection. On receiver side, opposite process was applied to get the character by *dividing* the sequence number's value on $(65536 * 256)$. The big disadvantage, when every character is transferred through a connection, is that many requests were made to connect to the server without receiving Synchronize- Acknowledgement (SYN/ ACK) packet would attract the attention.

Ciobanu [CIO06] suggested SCONEP (Steganography and Cryptography Over Network Protocols) and used ISN after solved the issue which was appeared in **[ROW97]** by sending a Reset (RST) packet to abort a connection instead of an Acknowledgement (ACK) packet after 4-bytes would be transmitted. The data was encrypted and compressed before transferred.

Singh [SIN13] implemented data hiding by using the identification field (16-bit) with ISN field (32-bit) for disguising (6-bytes) of characters after encrypted it using an algorithm which was chosen by the sender and whether to compress it or not is determined by the sender.

Biswas [BIS16] was using the sequence number field as a carrier for Rivest-

Shamir-Adleman /Data Encryption Standards (RSA / DES) key that was used to encrypt the data then, the ciphertext was embedded in the data field. The receiving packet was captured through wireshark application. After the ciphertext was taken from the data field and obtaining the key from the sequence number field, the ciphertext was decrypted to get the data.

1.3 Aim of Thesis

The aim of thesis is hiding the data by using field of TCP header. It is implemented by constructing the packet that consists of headers and payload. The sequence number field of TCP header was chosen for performing data hiding and sending it to network traffic. The intended recipient is captured the constructed packet and extracted it to retrieve the original data.

1.4 Thesis Layout

The remaining part of thesis was including four chapters as follows :

- **Chapter Two:** focuses on viewing TCP / IP model with their protocols especially IP and TCP protocols and their headers, viewing the client – server architecture in a brief manner and sockets. It was also viewing information hiding and turning to steganography concept historically and recently. It is also displaying Linux operating system that is considered the environment of work.
- **Chapter Three:** it is over viewed how to implement steganography using IP

and TCP headers fields. It is also viewing a layout of system model, the proposed methods required, and the functions for performing the work.

- **Chapter Four:** it is viewing the results of the implemented work.

- **Chapter Five:** it is viewing the discussion, the most prominent conclusions about over all work and the future work.

CHAPTER TWO

CHAPTER TWO

TCP / IP AND STEGANOGRAPHY UNDER LINUX ENVIRONMENT

2.1 Introduction

The TCP/IP Reference Model and its layers are defined in a brief manner. Internet Protocol (IP) and Transmission Control Protocol (TCP) are explained with its headers. This chapter is also viewed the concept of client- server Model and packet analysis. Because the Linux operating system is the environment, the description of this system is displayed.

2.2 TCP / IP Reference Model

The necessity for general model communicates hundreds of universities and government installations which were communicated firstly by rented telephone lines was appeared after satellite and radio networks appended to these communities because this due to troubling interworking of existing protocols. So U.S. Department of Defense (DoD) was sponsored research network which was known as Advanced Research Projects Agency Network (ARPANET) that time and the main goal of it was to override connectionless between connected multiple networks. Later this model became known as TCP/IP Reference Model [TAN11].

Functions of data communications protocols were described by an architectural model which was known as Open Systems Interconnect (OSI) Reference Model that developed by International Standards Organization (ISO). These functions were defined by seven layers that comprised in OSI Reference Model.

As mentioned, functions of data communications may be preceded by any number of protocols so, layer did not define a single protocol but multiple as needed by these functions. For example File Transfer Protocol (FTP) and Simple Mail Transfer Protocol (SMTP) are two protocols for different user services that existed in Application layer.

OSI layers were helping to give description to TCP /IP with layered model. Definition of layers for this model between three to five layers as functional levels represented description for TCP / IP. This model consists of four layers where each layer has its own functions and protocols as illustrated in Figure (2.1):

1. **Link Layer or Network Access Layer:** It is performed in a network adapter that sometimes was known as NIC. The basic function of this layer is to transfer datagram (explained later) from one node to another. Also there is some other functions include framing, link access, reliable delivery and error detection and correction [KUR13]. The technologies are used in this layer include Ethernet (Local Area Network Technology). In Ethernet, a shared link is used for sending and receiving by set of nodes. Token ring is another technology. It simply means the set of nodes connected as a ring. Frame Relay and Asynchronous Transfer Mode (ATM) are examples of virtual circuit technologies. It requires setting up a connection before any data is sent [PET03].
2. **Internet Layer:** The current Internet is using IP version 4 (IPv4). The main functions of this layer are addressing, routing and fragmentation. Internet protocol (IP) represents major protocol of this layer. In addition to IP, Address Resolution protocol (ARP), Internet Control Message Protocol (ICMP), and

3. Internet Group Management Protocol (IGMP) also exist. Host – to – host represents the property of connection [HUN02].
4. **Transport Layer:** the location of this layer is above internet layer. It mainly provides end- to – end communication which means that the process on a source host takes a message and deliver it to the process that runs on a destination host. Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are protocols that supplied by this layer.
5. **Application Layer:** it is the top layer of TCP / IP model. The process requests from hosts and ensuring that connection is turned out to suitable ports is made by this layer [BEA09]. Many of application protocols are included in this layer. Type of these applications is different between providing user services and system administration. Telnet, SMTP, FTP, and Hyper Text Transfer Protocol (HTTP) are used as user services while Domain Name System (DNS), Open Shortest Path First (OSPF), and Network File System (NFS) are used as both user services and system administration [HUN02]. In addition to these protocols, Simple Network Management Protocol (SNMP) operates on this layer. It represents the standard of TCP/IP for network management.

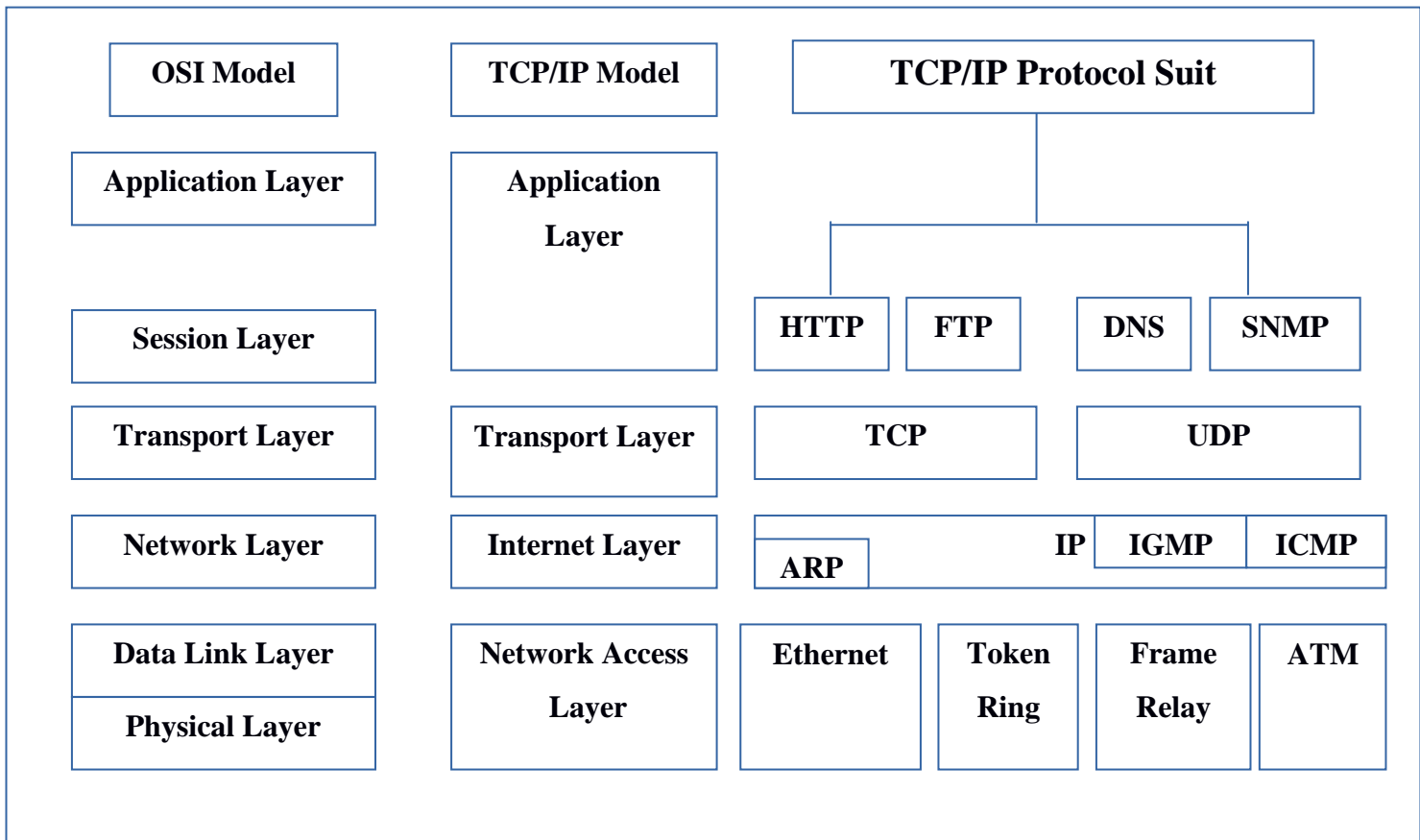


Fig (2.1) TCP / IP Layers and Protocols

When the data was being sent to a network, it should be passed down the stack while its receiving should be passed up. TCP / IP four layered structure could be seen as the data was passed down from Application layer. Control information related to each layer would be added to the data through passing layer for ensuring the suitable delivery. This control information was added in front of the data to be transferred and it was known as *a header*.

Each layer was considering the information that came from above layer as a data and placed its own header before it. Adding of these control information was known as **encapsulation** process. On receiving, the opposite process happened. Before the data was passing to upper layer, each layer removed its own header since

information was received; it was interpreted as header and data. This process was known as **decapsulation** as illustrated in Figure (2.2).

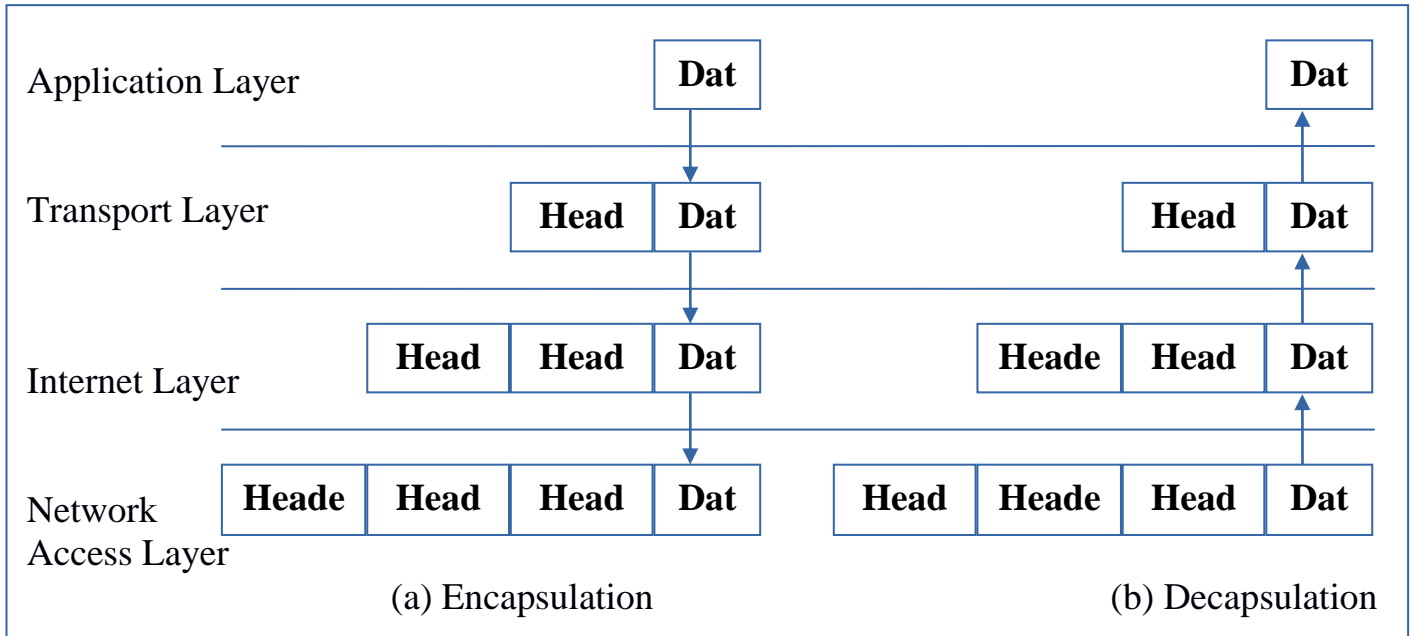


Fig. (2.2) Encapsulation and Decapsulation

The different terms were applied to different layers for expressing the data to be transferred. Stream was known to the data that was used by TCP's Applications while message was known to the data that was used by UDP's Applications. In transport layer, TCP's data was named segment while UDP's data was packet. Datagram represents the name of the data whether it was segment or packet.

The different underlying networks were using different terminology for expressing the transmitted data [HUN02].

This model was also known as TCP/IP Protocol Suite. It clearly seems that protocol suite consists of many protocols but its name came especially from the two main protocols TCP and IP. TCP was a reliable connection oriented protocol, which will be explained later while IP was a connectionless.

2.2.1 Internet Protocol (IP)

It was considered the major protocol in Internet layer. It was responsible for providing addressing and routing globally. So, it supports universal connectivity. Also, it supports fragmentation [DOR16]. The most important characteristic is that it was best effort or connectionless. The meaning of best effort is that the datagram could be lost, corrupted or reaching out of order [FOR07] while connectionless means that before transmitting the data, there was no exchanging control information (known as handshake) to establish end – to end connection that was provided by connection oriented protocol for example TCP. So, it did not provide acknowledgment or retransmission when packet loosed or out of order. Because connection oriented was required, IP must depend on protocols of upper layer that provided it [HUN02].

As mentioned previously, the data defined as datagram that means different routes would followed by datagram through transferring from source to destination. IP is known as IPv4 because (4) is the current version.

Datagram consists of header and payload. The length of header was variable and ranges between 20-60 bytes proportion to the existence of option field. This variation results in variable length datagram. The header of IPv4 is shown in Figure (2.3) [FOR07] includes:

- **VER (version):** it is a 4-bit length. It indicates the format of IP datagram that is IPv4.
- **IHL:** this field specifies IPv4 Header Length in 4-bit. Its value either 5 words (20 bytes) that are representing default value if there is no option field or 15 words (60 bytes) otherwise.
- **Type of Service or DS (Differentiated Services):** it is an 8-bit length and over

years it took different definition but the most famous is Type of Service (TOS). It is mainly representing how the packets would be processed through transferring component networks according to the necessity of application such as priority of packet.

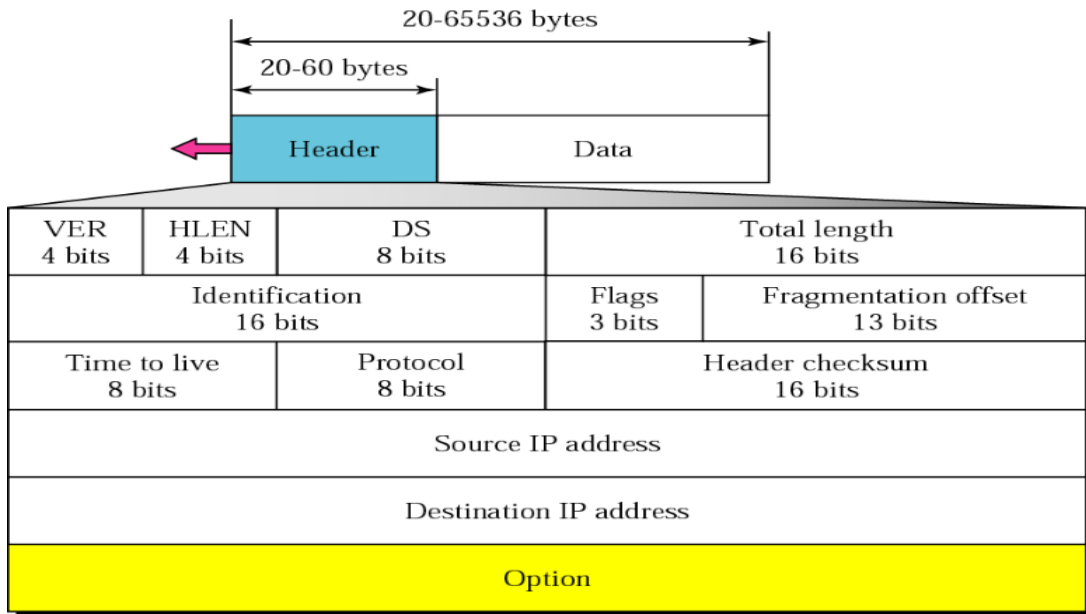


Fig. (2.3) IP header format [FOR07]

- **Total Length:** it is a 16-bit length and represents the length of datagram. Since it is 16-bit so, the maximum size of datagram would be 65,535. This seems to be too large to encapsulate by physical network due to fragmentation appearance.
- **Identification Field:** it is a 16-bit and it is mainly used in fragmentation process to identify fragments of one packet from other since each fragment belongs to the same packet has the identification field value incremented by 1.
- **Flags Field:** it is a 3-bit length. This 3-bit is interpreted as first bit is reserved, second bit Do not Fragment (DF) which means do not fragment datagram but datagram will be discarded if this is set and its size is larger than Maximum

Transfer Unit (MTU), and last bit is More Fragment (MF) bit. When it is set, this means the fragment is part of the original and every MF's fragment is set except the last.

- **Fragment Offset:** it is a 13-bit length and it is assigned to every fragment inside one datagram. Since it is 13-bit, so there exist 8192 fragments per datagram.
- **Time-to-Live:** it is an 8-bit length. This field's value is helping to prevent packet from routing loops by assigning it an initial value. This value is decremented by 1 at each router until it is reached to 0[FOR07].
- **Protocol:** it is an 8-bit length and its value identifies protocol of above level that carried by IP datagram. This helps the receiver when decapsulate the received packet to decide higher level protocol should deliver its payload that included next header of upper layer. The value of this field is 6 for TCP, 17 for UDP and 1 for ICMP.
- **Header Checksum:** it is a 16-bit length and its benefit to detect error of the datagram. The value must be updated after router modifies Time to Live field when it decremented it and fragmentation fields if fragmentation happened. To calculate checksum of header, one's complement to result of using one's complement arithmetic to adding up 16-bit half words of header. So when datagram is arrived a router, the router compare computed checksum with the one of the received packet. If it differs, the datagram must be discarded because header is corrupted [TAN11].
- **Source Address:** it is a 32-bit length and is clearly representing source IPv4 address where datagram is created. It refers to network interface not to host. It is represented as a dotted decimal notation that means each byte of 4-bytes takes values in a decimal between 0-255[TAN11].

- **Destination Address:** it is a 32-bit length and it is representing intended destination IPv4 address where the packet must be reached.
- **Options:** its role could be summed up including the information that does not exist in the original design. Its length is variable. Such options like security, Strict source routing, loose source routing, Record route and Timestamps. Today option fields scarcely used [TAN11].

2.2.2 Transmission Control Protocol (TCP)

TCP is a connection oriented protocol, meaning that the data must be transferred after the connection is established. In addition to establish connection and data transmission, the connection must be released. Because it uses sequence number (described later) to ensure delivery correctness and confirmed no data was lost when network failure occur by applying retransmission / timeout mechanism, so it was considered a reliable protocol. It is also using sliding window algorithms to transfer large files. Also, it is stream-orientation because it uses buffer in sending and receiving and this enable application to write very small or an amount of data and divide it into appropriate size [DOR16]. It is also process – to – process communication through using port numbers [FOR10]. Port numbers from (1- 1023) are system ports. Port numbers from (1024- 49151) are registered ports while ports from (49152-65535) are private ports.

TCP segment consists of header part and data part. As shown in Figure (2.4), the header part fields are:

- **Source Port Address:** it is a (16-bit) length and its role is identifying sending service in.
- **Destination Port Address:** it is a (16-bit) length and its role is identifying receiving service from.

- **Sequence Number:** it is a (32-bit) and it is assigned to every byte of TCP segment since TCP is byte stream as mentioned previously. When the sequence number must be generated randomly through establishing connection, SYN bit flag should be set and it was known as ISN.
- **Acknowledgment Number:** it is also a (32-bit) and it is essentially representing the sequence number that expected to receive as a next data byte. ACK bit should be set with this field **[FOR10][MAR13]**.

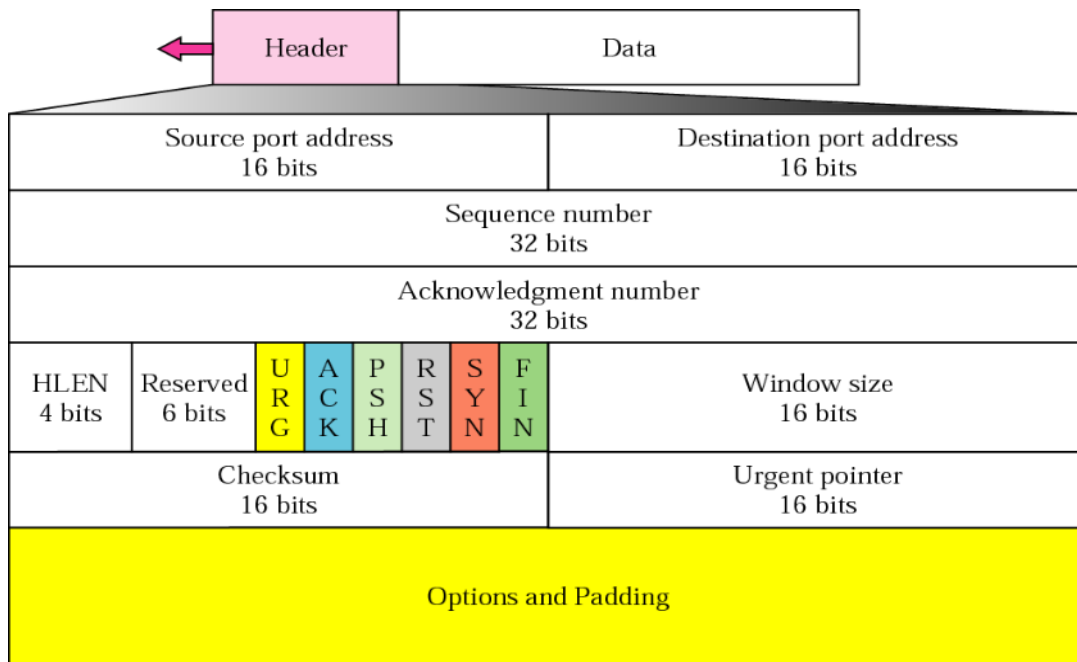


Fig. (2.4) TCP header format **[FOR07]**

- **HLEN:** also known as offset field. It determines TCP header length as 32-bit words. Since header length was variable depending on option field existence so, the regular value would be 5 words (20 byte) if option field does not exist while it is 15 (40 byte) with existence **[MAR13]**.
- **Reserved:** it is also known as unused field. It is set to 0 since it is specified for using in future.
- **Flags :** six flags are contained in this field as follows :

- **URG (Urgent):** describes the data of urgent pointer. It is validated when it is set
- **ACK (Acknowledgment) :** this field tells recipient to take attention of acknowledgement number since it is set when the field of acknowledgement number of header is valid
- **PSH (Push):** when it is set, the received data must be passed to the receiving application by TCP receiver.
- **RST (Reset):** a setting of this field means that the receiver must abort a connection because some conditions are abnormal. For example when the received segment is not expected by the sender so the connection is aborted like port scanning that is performed by an attacker.
- **SYN (Synchronize):** the connection establishment required setting this bit.
- **FIN (Finish):** when TCP sender does not have any data to send to TCP receiver, this bit must be set to inform that. Although TCP sender has no data to be sent, it can receive a data from TCP receiver until it sends segment with FIN bit setting.
- **Window:** it specifies the number of bytes that the sender can accept when it represents as a TCP receiver. This important to control flow of data and congestion.
- **Header Checksum:** this field is useful to detect errors in the receiving segment.
- **Urgent Pointer:** the value of this field should be added to the sequence number value when URG- bit is set. The data in the received buffer should be considered urgent when it is pointed by this field.
- **Option and padding:** the functions that do not cover by a regular TCP header

may be provided through this field. The maximum length of this field is 40-byte and extra padding bits must be added when its length is not a multiple 32-bit [MAR13].

2.2.3 Packet Analysis

Packet analysis, often named as a packet sniffing or protocol analysis, symbolizes to the capturing and interpreting process of live data when it flows via a network. The tool was known packet sniffer was used to capture a raw network data via the wire.

Packet analysis could help to understand the characteristics of network, studying who is on a network, the determination of peak network usage times, the determination of malicious activity or possible attacks, and finding out unsecured applications.

The packet-sniffing process included three steps [SAN11]:

1. Collection: the purpose of this step is collecting a binary data from transmission media whether it was wire or wireless.
2. Conversion: after collecting the binary data, it must be converted to a legible form.
3. Analysis: this is the final step where analyzing the captured and converted data.

2.3 Client- Server Architecture

This architecture was defining two hosts. One of them was services allocation known as a server and the other represents all hosts that request services known as

clients. The web application represents an example for this architecture where browser (client) sends requests to web server (server). Actually these services could be considered as processes (program in running) and exchanged between two hosts as messages. These messages must be passed through the underlying network in sending and receiving. So to perform this process, a software interface (socket) was needed [KUR13].

2.3.1 Application Programming Interface

Socket or Application Programming Interface (API) represents the important interface in writing network applications. Its position between Application layer and Transport layer as shown in Figure (2.5) so the application developer would be controlled application layer widely while controlling transport layer was little. The type of the used protocol and maximum buffer should be received could be determined by an application developer [KUR13].

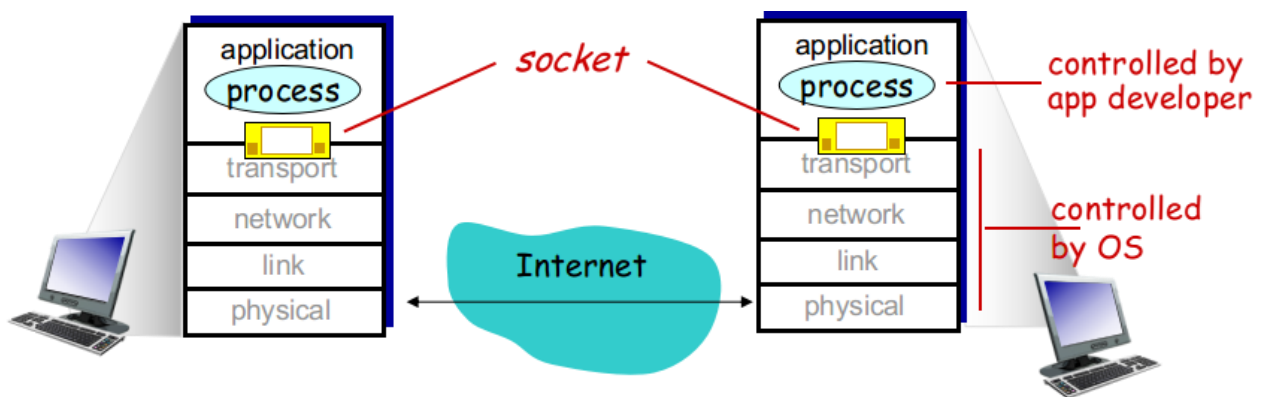


Fig.(2.5) socket [KUR14]

Sockets could be classified according to the properties of communication apparent to user. For processes communication, sockets of the same type were

supposed. There were four types of sockets as follows [LEF86]:

1. Stream socket: this type was provided for bidirectional, reliable, sequenced, and unduplicated flow of data without record boundaries.
2. Datagram socket: this type was provided only for bidirectional but not for sequenced, reliable, or unduplicated data flow.
3. Raw socket: this type was provided for accessing underlying communication protocol. It was datagram-oriented and mainly dependent on interface provided by protocol. It was not specified for general user but for ones that concerned in accessing some facilities that esoteric of an existing protocol or development new communication protocols.
4. Sequenced packet socket: it was a part that provided in Network systems (NS) socket abstraction. It was similar to stream socket except that preservation of record boundaries.

Figure (2.6) states the relation for each socket with protocol related to it. The most important property of a raw socket that was allowing new protocols of IPv4 to be performed in user space. Sending or receiving raw datagram without header of link layer was provided by raw socket [KER07].

There are many types of socket's functions. The socket function that was used for creating socket was `socket` while the others that were used in connection included `connect`, `bind`, `listen`, and `accept`. Another type of functions was used for sending the data over socket included `send` and `sendto` while `recv` and `recvfrom` were used for receiving data from socket [KER07].

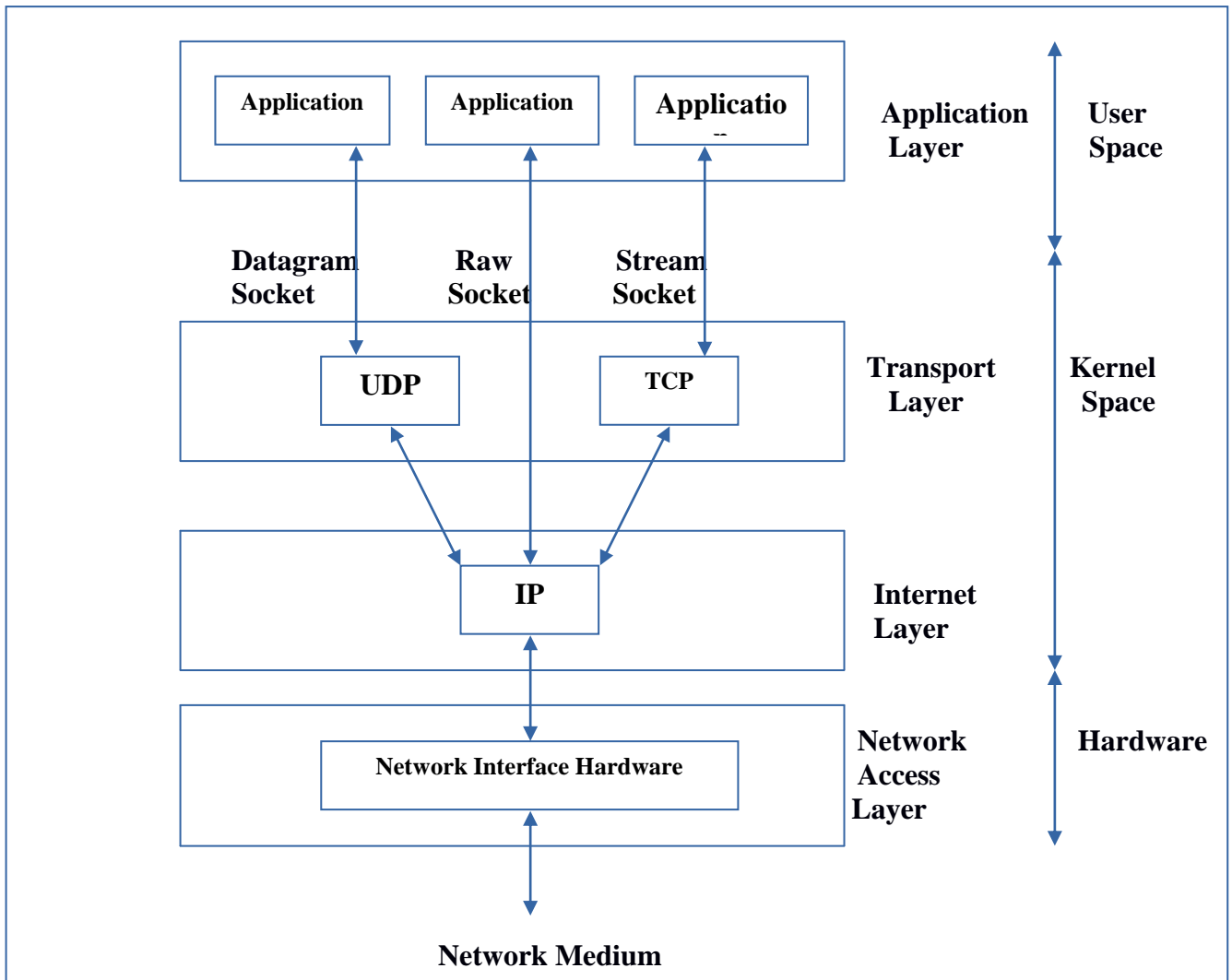


Fig. (2.6) Types of Sockets

Figure (2.7) explains TCP client – server connection. It clearly seems that server's Listen function is waiting for a request from a client through connect() operation. After server accepted client's request, the connection is established and data will be transferred.

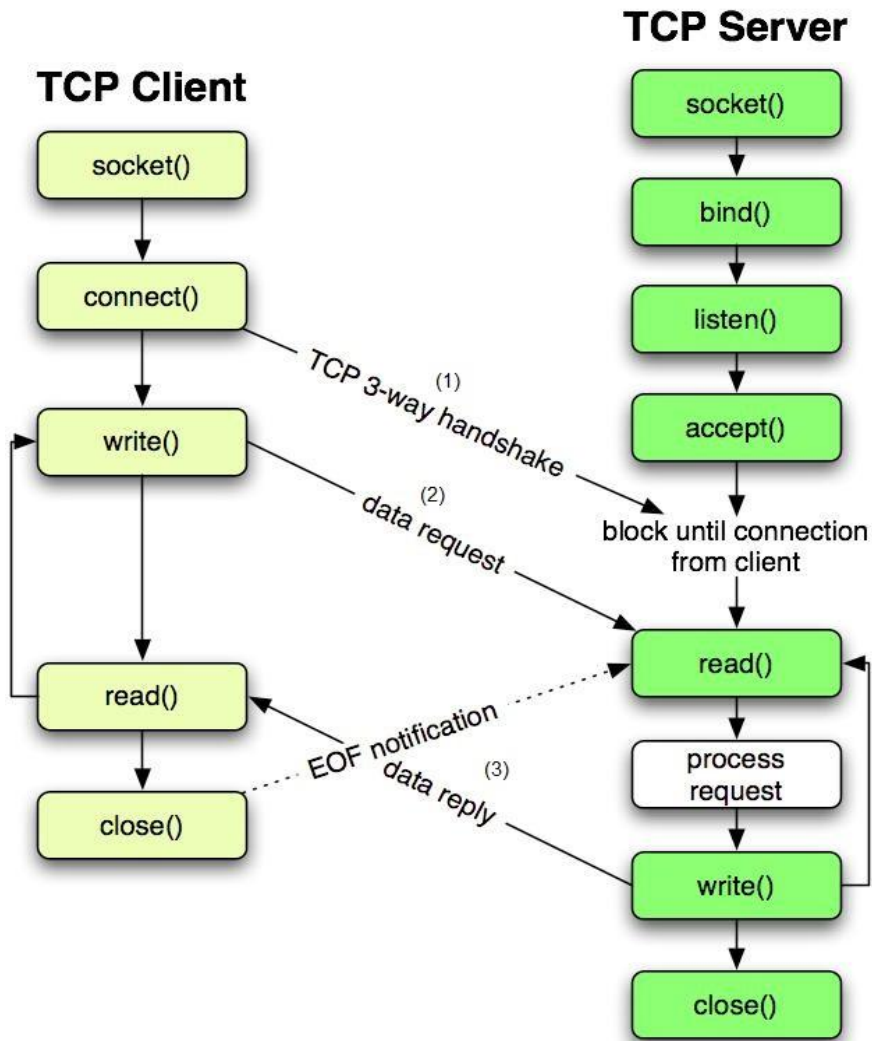


Fig. (2.7) Client – Server connection through TCP Socket [PER05]

a. TCP Establishment Connection

Before TCP establishment connection explained, types of exchanged messages must be mentioned. Three types of messages were using three abbreviations with respect to TCP header flags. These are:

- **SYN (A Synchronize message):** an initiating and establishing a connection happened when this bit set. Because one of its functions was to synchronize sequence numbers, so it was named as SYN message.

- **FIN (A Finish message):** when a device wants to terminating a connection, the FIN bit set.
- **ACK (An Acknowledgment message):** a reception of a message such as a SYN or a FIN was indicated when this bit set.

The establishing connection between a TCP client and server included the following three steps as shown in Figure (2.8):

- A SYN message was sent by client.
- A message was sent from server combining an ACK for the client's SYN and contains the server's SYN.
- Finally, an ACK sent from client for the server's SYN.

SYN and ACK messages used for establishing a connection. The Initialized connection from segment indicated by SYN. SYN represents synchronize that refers to sequence number synchronization duty. The device that sending a segment is transferring an acknowledgment when it is received a message.

Sequence of steps that was taken in a TCP session could be represented as Finite State Machine (FSM). FSM is a theoretical tool used for a protocol description. The concepts of FSM are as follows:

1. State: the protocol software on a machine is described through this status at a given time.
2. Transition: the moving from one state to another is representing this concept.
3. Event: a transition between states is occurred by something causes that.
4. Action: before device transitions to another state, it did something for response to an event.

All the different states the protocol could be in, the events that could be happen, the

actions were taken for responsive to the events and the transition occurs as a result are explained by an FSM.

Table (2.1) describes each state and its description of the client and server in the establishment of connection.

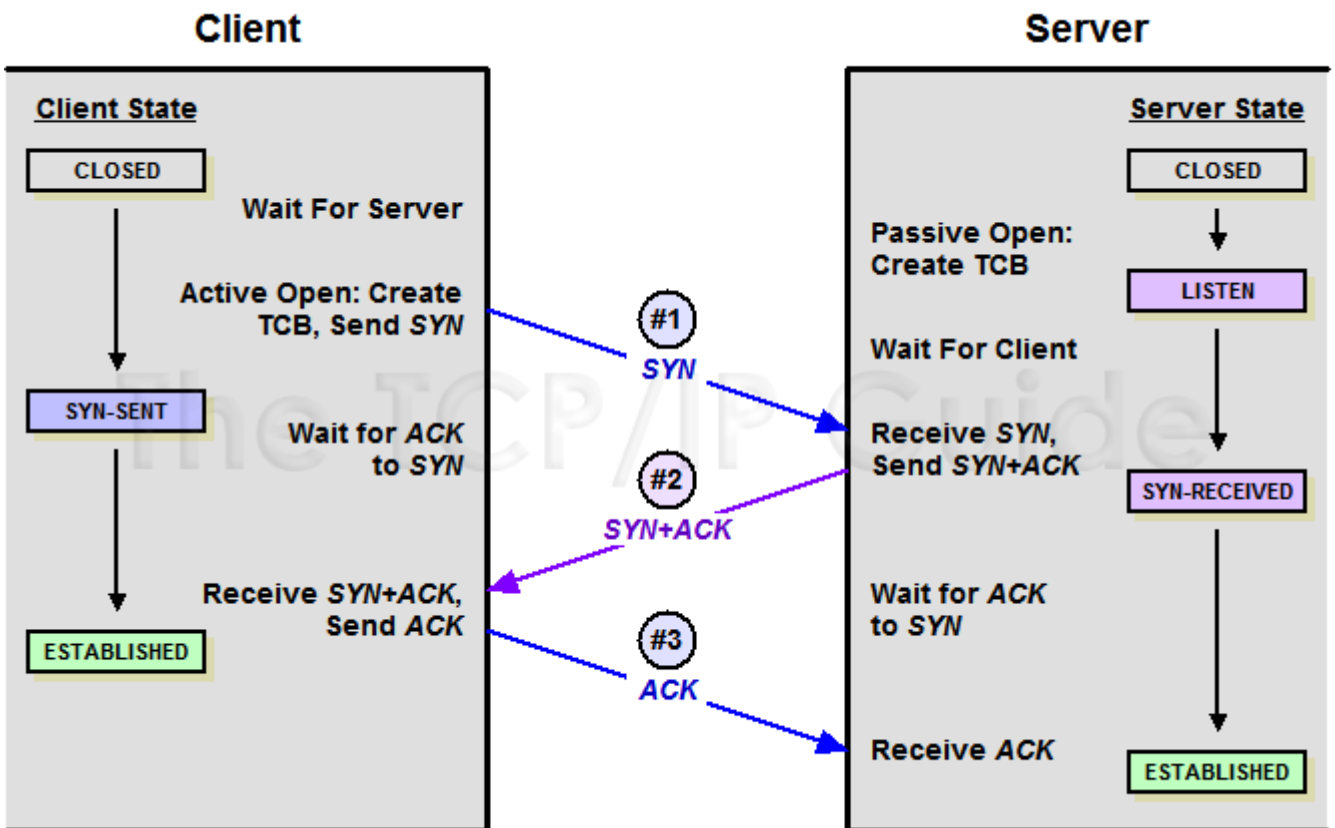


Fig. (2.8) three-way handshake

Table (2.1) states and state's description

State	State Description
CLOSED	Each connection started with the default state (CLOSED) before connection established
LISTEN	Related to device that waiting to receive SYN message (server)

SYN-SENT	Related to device that sending SYN message (client)
SYN-RECEIVED	Related to device that receiving SYN message (server) and waiting for ACK message after sending its own SYN
ESTABLISHED	Means TCP connection was opened

Segment with the RST bit was created by the device that wants to reset connection so that the connection could be reestablished. To ensure that the action is valid, the value of sequence number field was checked to guarantee that it belongs to the reset itself. The handling of message when reset is valid depends on device's state that it receives as follows [KOZ05]:

- If the state of device is LISTEN, the reset is ignored.
- If the state of device is SYN-RECEIVED but LISTEN was the previously state, it will return to the LISTEN state.
- The device returns to the CLOSED state for that connection because the reset aborted the device connection.

2.3.2 Server

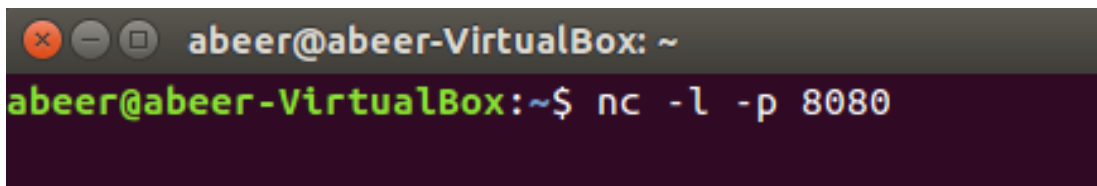
Server model can be programmed or using tools. netcat (also known as nc) represents Unix utility that is dealing with reading and writing data through the connection of network by using protocols of transport layer.

It is a reliable tool either used by other programs or scripts or used directly. Netcat can be used as a server by doing reading or writing for inbound connections on arbitrary ports by listening [FRY11].

There are many options that is used by nc but two options are concerned:

1. **-l**: listening for an incoming connection by nc and not initiate connection.
2. **-p source_port** : nc should use source port specified by this option[HOB].

nc command executed through terminal shown in Figure (2.9)



```
abeer@abeer-VirtualBox: ~
abeer@abeer-VirtualBox:~$ nc -l -p 8080
```

Fig (2.9) nc command

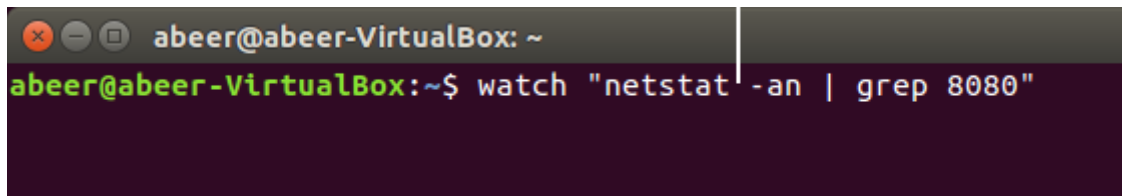
Another tool which is used in conjunction with nc that is **netstat**. Netstat ("network statistics") have many advantages like displaying network connections whether it is incoming or outgoing, routing tables and a number of network interface and network protocol statistics. It is available in many operating systems like Linux, Windows, etc. Finding problems in the network and the performance measurements by determining the amount of traffic on the network are implemented through this tool.

Netstat like nc uses many options but two options of them are concerned:

- **-n**: instead of determining symbolic host and port, the numerical addresses are showing.
- **-a, --all** : showing both listening and non-listening sockets through this option.

All active connections to the server are shown through **netstat -an** [BAU13].

Running command repeatedly and displaying of output is performed via **watch**. It has ability to change watching program output over time. By default, every two seconds, program is executed [COL99]. Watch command used with netstat as shown in Figure (2.10)



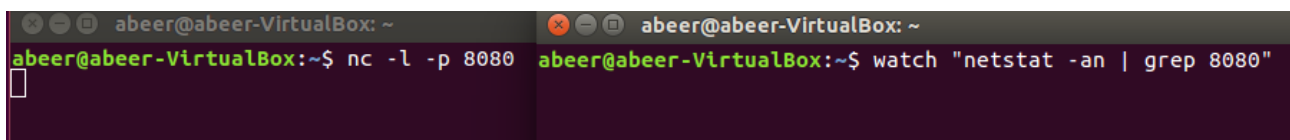
```

abeer@abeer-VirtualBox: ~
abeer@abeer-VirtualBox:~$ watch "netstat -an | grep 8080"

```

Fig (2.10) watch with netstat commands

the terminals with nc and watch with netstat shown in Figure (2.11)



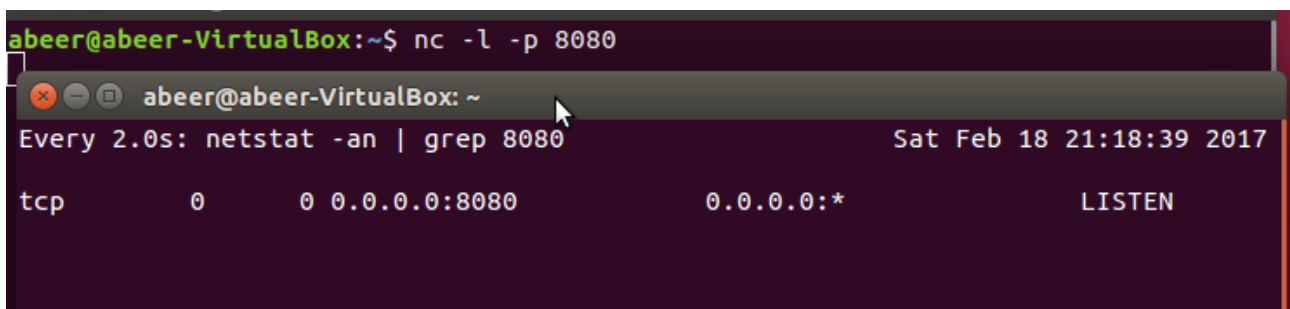
```

abeer@abeer-VirtualBox:~$ nc -l -p 8080
abeer@abeer-VirtualBox:~$ watch "netstat -an | grep 8080"

```

Fig. (2.11) nc with watch and netstat

After executing watch with netstat commands, it will seem as shown in Figure (2.12)



```

abeer@abeer-VirtualBox:~$ nc -l -p 8080
Every 2.0s: netstat -an | grep 8080                               Sat Feb 18 21:18:39 2017
tcp        0      0 0.0.0.0:8080          0.0.0.0:*            LISTEN

```

Fig. (2.12) watch with netstat commands after executing

2.4 Information Hiding

Information or data is an important resource. After the communication was developed through the emergence of network, the information or data was transferred through communication media. Transmission means do not provide any security to protect it from unintended recipient through transmission so that, finding methods for

protecting it became very necessary.

Many types of data required protection such as personal and private data, sensitive data, trades secret and confidential data, avoiding misapplied data... etc[GUP14].

Many ideas were defined to protect it. Some of these were depending on hiding the content of data (cryptography) while the others were depending on concealment the existence of it (steganography). In this thesis, steganography was considered direction for hiding information but it was necessary to view the meaning of steganography historically and recently.

2.4.1 History of Steganography

Previously, especially in ancient Greece, Herodotus (c. 486-425 B.C.) was told about how a message was sent for inciting mutiny against the Persians. So to guarantee that anyone could not notice that, Histiaëus chose the most faithful slave and shaved his head, tattooed it with the required message and waiting until his hair had regrown, then he was sent[KAT00].

2.4.2 Modern Steganography

Recently, with computer and information technology emergence, information hiding takes another direction by using text, image, audio and video which were representing a suitable carrier for transferring secret information. Also, Protocols of TCP / IP protocol suite can be utilized to transfer these information which was known Network Steganography protocol [SIN13]. It was introduced firstly by Krzysztof Szczypiorski in 2003 through implementing hidden data in Hidden Communication

System for Corrupted Networks (HICCUPS) [SZC03]. Before that, the concept that was known as covert channel produced by Lampson in 1973 which provides desirable environment. It means utilizing unused fields of network protocols or changing uncritical data [ROH11]. This leads to create a carrier to hide information far away protocol specification. Because the requirement of steganography for carrier to hide information, the network steganography can be implemented through existence of covert channel [CIO06].

There were many types of covert channel depending on what was using as a carrier:

1. Storage Covert Channel: a shared resource was used by one process to write and was read by other process [MIL14].
2. Timing Covert Channel: in this channel, an event's timing was used for covering information.
3. Hybrid Covert Channel: a combination of the two previous channels was used.

Network steganography methods could be intra - protocol or inter- protocol where intra- protocol was using a single protocol to perform the hidden communication while inter-protocol was using two or more protocols to hide information [JAN10].

Intra – protocol methods could be classified depending on which part was exploited as described in covert channel above to:

1. Methods that are modifying protocol structure whether it was payload, headers or both for example IP, TCP, UDP headers.
2. Methods that are modifying relations of time between Protocol Data Unit by modifying delay or order of packets
3. Methods that are modifying both structure and time relations [MAZ13].

It was obvious whether in historical or modern steganography, this process

needed essential elements for hiding as follows:

1. Carrier: also known as cover- object and defined as the place where a message was included and concealed existence of it.
2. Message: represented the data which the sender wants to be secret.
3. Password: also known as a stego – key that represented a decoding key which was known only by the recipient and hence would be extracting the hidden data from the cover object [AMI03].

Analysis of these covert channel could be performed by calculation the number of bits (steganogram) transferred during a packet according to the equation supposed by

$$\text{[MAZ08]} : PRBR_{NS} = \frac{(SB_0 + \sum_{j=1}^l SB_j)}{l+1} [\text{bits}/\text{packet}] \dots \dots \dots (2.1)$$

where:

PRBRNS (Packet Raw Bit Rate) denotes the bandwidth of the covert channel created by IP/TCP/UDP steganography [bits/packet],

SB0 is the total amount of bits for IP/TCP/UDP protocols that can be covertly sent in the fields of the first packet. This value differs from the value that achieved for the following packets because in the first packet the initial values of certain fields can be used (e.g. sequence number for TCP protocol),

SBj denotes the total amount of bits for IP/TCP/UDP protocols that can be covertly sent in the fields of the following packets, and

l is the number of packets that sent besides the first packet.

2.5 Linux

Linux was a copy of UNIX operating system. Personal computers represented with Intel 80386 one of variety of platforms that Linux run on. A wide range of software was supported like GNU C/C++ compiler, TCP/IP. Multitasking and multiuser were properties of Linux as same as the other versions of UNIX.

Ext2 file system was supported by Linux as various file systems supported by it that is used to storing data. Advanced features like the protected-mode that provides multitasking, descriptor based, and memory-management paradigm was used by Linux after developing it. Disk paging was implemented by Linux for increasing the amount of memory.

Dynamically linked and shared libraries were used by executable files which occupy a little disk space and supported by Linux. Routines that implemented by programmer could be used instead of standard ones.

The standard libraries, programming tools, compilers, and debuggers that embedded in UNIX programming environment which provided by Linux.

TCP/IP networking software also provided by Linux and many Ethernet cards and interfaces also provided. Also accessing Internet through modem was provided by supporting Serial Line Internet Protocol (SLIP) and Point-to-Point Protocol (PPP) [WEL96].

CHAPTER THREE

CHAPTER THREE

STEGANOGRAPHY IN TCP / IP SYSTEM DESIGN AND IMPLEMENTATION

3.1 Introduction

The proposed system model is viewed in this chapter. All the proposed methods for three-way handshake and generation of a sequence number are displayed. Sending request (SYN packet) and replying (SYN-ACK packet) that represents the two steps of three-way handshake are implemented. The two proposed methods for hiding text in a sequence number field are executed. The two proposed methods for extracting the hidden data are viewed too.

3.2 Text Hiding Using Sequence Number field Implementation

1. Although, there are fields that are suitable for hiding like reserved or option fields of TCP header protocol but sequence number was chosen for hiding information. A reserved field is not used because it is designed for future use. All bits set to zero so any change to its values may attract attention. Although the option field size up to (40- byte) but it is susceptible to filtering. This leads to exclude from hiding.
2. The choice of using sequence number field means that TCP and IP headers must be created manually because in the normal transmission of data in client – server architecture, operating system's kernel was taking care of adding required headers.
3. As mentioned in chapter 2, IP and TCP header have variable size between 20-

60 bytes depending on the existence of an option field. In the proposed system, the option field was not taken into consideration.

4. Some fields of the two protocol headers must remain unchanged through data transmission while others must be changed.
5. The unchanged fields of IP protocol header are version, header length, and type of service. Other fields like identification, flags, and fragment offset that specified for fragmentation strategy are changed through fragmentation, total length, time to live, checksum, source address and destination address will be changed. Protocol field has (6) value that is pointed to payload of IP header is TCP segment.
6. In other side, fields of TCP header like source port, destination port, sequence number, acknowledge number, flags, checksum, and window fields should be changed through transmission.
7. Regardless of which operating system was used, root privilege level access should be used since custom header of packet was created.
8. As mentioned earlier, through packet creation, the sequence number field was used in hiding data. In the proposed system, two methods for generating and hiding data in sequence number field were used.
 - i. In the first one, source and destination ports of TCP header fields were used.
 - ii. A combination of fields from IP and TCP protocols headers (source and destination ports, version, and protocol) were used to generate the sequence number field in the second method.
9. Sending packet that included the hidden data in sequence number field was in SYN packet. As described earlier, SYN packet is the first packet in three way handshake process.

10. Receiving packet to extract the hidden data was in SYN-ACK packet which represents the second packet in three way handshakes and also acts as a reply to the connection request from sender.

3.3 System Model

The structure of the proposed system for Hiding Text in TCP / IP is illustrated in Figure (3.1). It consists of three models. The sender (client), server and receiver are the three models respectively.

Three parts are in the sender model. These parts are: create packet, send SYN packet, receive SYN-ACK packet. The second model is the server. The main function of the server is to listen for the connection and reply to the sender. The last model is the receiver model. It consists of three parts. These parts are: capturing packet, analyzing packet and data extraction procedures.

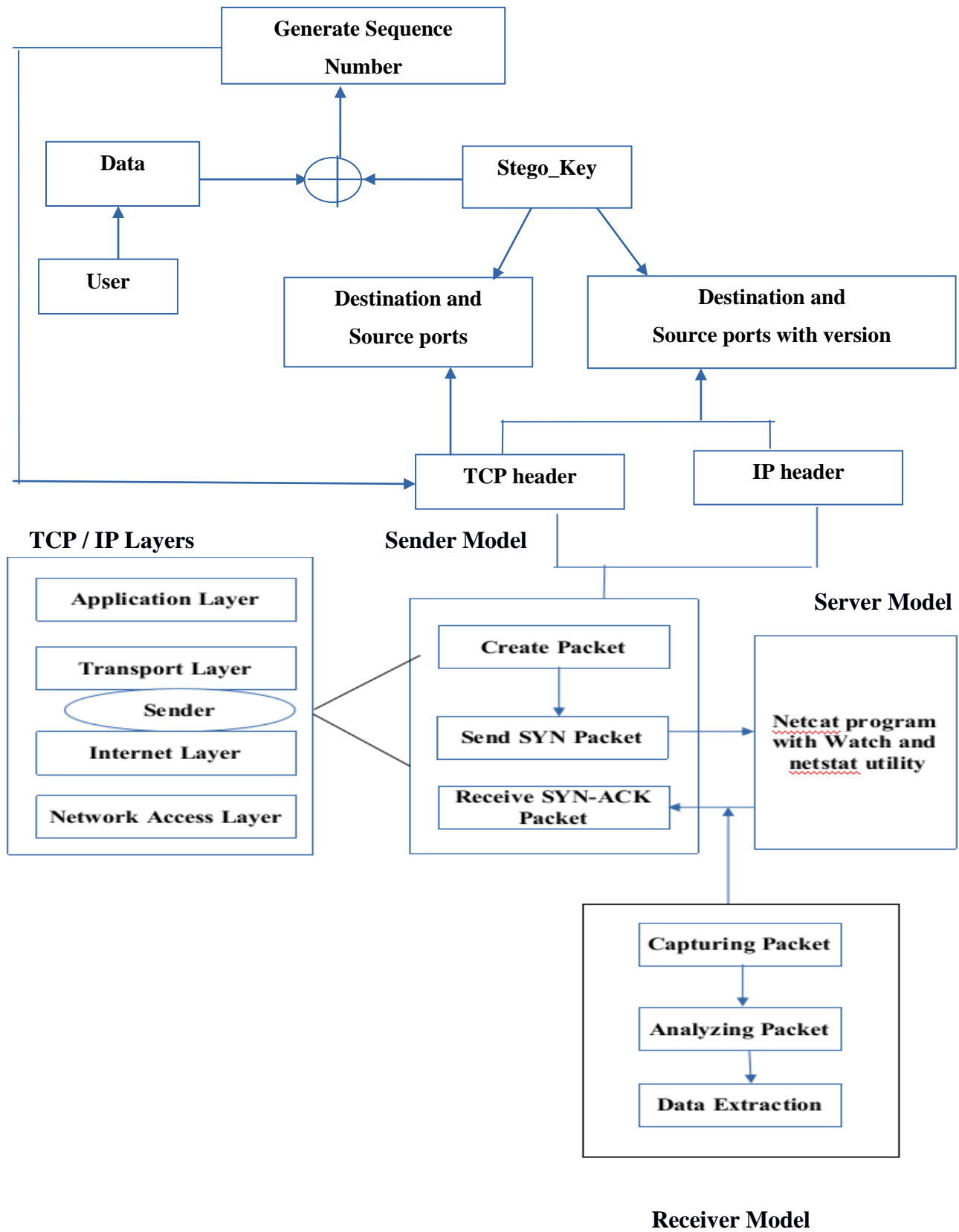


Fig. (3.1) the proposed system model

3.4 Sender Model

As described previously, the main parts of the sender model were generating a packet, sending SYN packet and receiving SYN-ACK packet. The description of these parts will be described in the next section.

3.4.1 Generate Packet

The process of creating IPHeader, TCPHeader, and the final packet to be sent to the server were the main functions of this module as described in algorithm (3.1). Each function in this module will be described separately in the next section.

Algorithm (3.1) Generate Packet

Input:

None

Output:

Packet consists of IPHeader + TCPHeader + Data

Begin

Step 1: calling createIPHeader function

Step 2: read data to be hidden

Step 3: calling createTCPHeader function

Step 4: create packet

Step 5: return packet

End

a. Create IPHeader Function

The process of creating the IPHeader was the main purpose of this function. In the beginning of this function, all fields of IPHeader were initialized. IP source and

destination addresses must be converted to network byte order. Version and IPHeaderLength(IHL) must be combined in one byte because each of them was 4-bit as follows :

$$\text{version_ihl} = \text{version} \ll 4 + \text{IHL} \dots\dots\dots (3.1)$$

and flags was 3-bit. It must be combined with offset as shown below

$$\text{flags_offset} = \text{flags} \ll 13 + \text{offset} \dots\dots\dots (3.2)$$

The pack function of struct library in python was used for gathering the fields of IP header together. Steps of this function are illustrated in algorithm (3.2)

Algorithm (3.2) CreateIPHeader

Input:
 None

Output:
 IPHeader

Begin

Step 1: set source and destination addresses as global variables

Step 2: set values to source and destination addresses

Step 3: pass source and destination addresses to IP class's object

Step 4: set result of grouping IPHeader's fields to iph variable

Step 5: return

End

b. Create TCPheader function

Like previous function, the purpose of this function is the same as the previous

one but the difference was to create TCPheader not IPHeader. In this function, the fields of TCPHeader were initialized.

Port number was selected randomly between (1024-65535). Destination port is chosen to be (8080).

As mentioned before, the carrier object for hiding data was the sequence number field. It must be random to prevent overlap in establishing connection. The generation of sequence number will be explained in the next section.

Since creating the packet were to establish connection, SYN bit field must be set. All the other control flags were set to 0. Steps of this function are illustrated in algorithm (3.3).

Algorithm (3.3) Create TCPheader

Input:

Data to be hidden

Output:

TCP header with hidden data

Begin

Step 1: set source and destination ports as global variables

Step 2: set result of random function to source port

Step 3: set value (8080) to destination port

Step 4: set value (empty string) to data

Step 5: pass source and destination ports to TCP class's object

Step 6: changing value of sequence number

Step 7: set SYN bit

Step 8: set length of data to data length field
 Step 9: set result of pack function (grouping TCPHeader fields) to tcph variable
 Step 10: return TCPHeader with the hidden data
 End

i. Generate Sequence Number

As described earlier, two methods are applied to hide data through the sequence number field. These methods are as follows:

1. Using TCP Header Fields

Source and destination ports were chosen to be a stego – key as shown in Figure (3.2). The generation of source port will be random through using random function in python.

A hiding is accomplished by using XOR operation. Since XOR is a logical operation, so all values were converted to binary. The steps of this function are illustrated in algorithm (3.4).

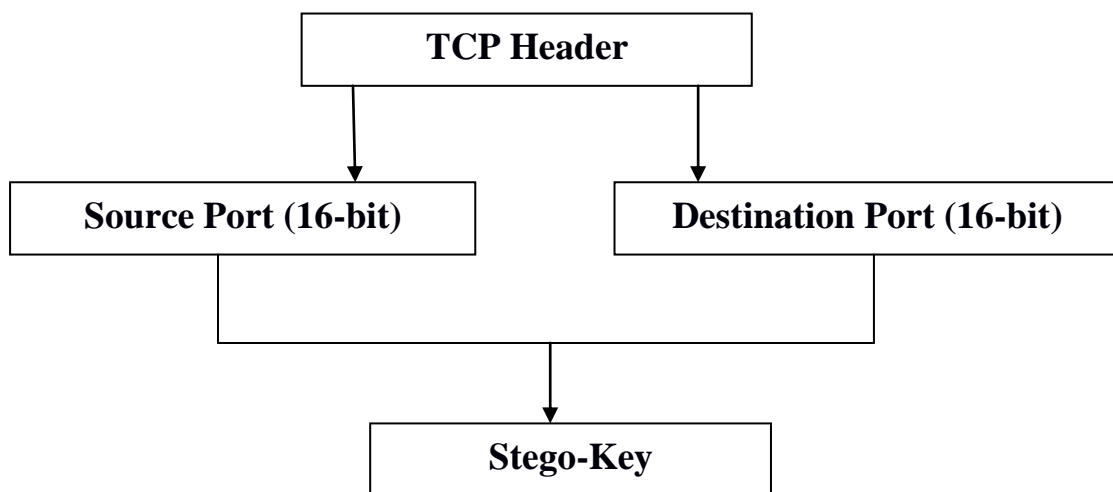


Fig. (3.2) Stego-key from TCP Header

Algorithm (3.4) Generate sequence number with srcdstports

Input:

Data to be hidden, source port, destination port

Output:

Sequence number with hidden data

Begin

Step 1: converting data's characters to decimal

Step 2: convert result of step1 to binary

Step 3: set result of step 2 to bindata variable

Step 4: converting source port to binary

Step 5: set result of step 4 to binsrcport variable

Step 6: converting destination port to binary

Step 7: set result of step 6 to bindestport variable

Step 8: concatenation source and destination ports in their binary representation

Step 9: set result of step 8 to binsrcdest variable

Step 10: implementation XOR operation between result of setp3 and step 9

Step 11: set result of step 10 to binsteg variable

Step 12: converting binsteg to decimal

Step 13: set result of step 12 to sequence number

Step 14: return sequence number

End

2. Using IP and TCP Header Fields

This method is the same as the previous one except that the destination port will be altered. Version and protocol fields from

IPHeader were selected for this alternation. This addition will make steganography process more complicated. The alternation of destination port will be on the second 8-bit as shown in Figure (3.3).

By returning to the IPHeader structure, an important thing can be noticed. The number of bits of version field is 4-bit and protocol is 8-bit. So protocol field will also be altered by replacing second 4-bit of protocol field value with version field value. Steps in algorithm (3.5) are illustrated a generation clearly.

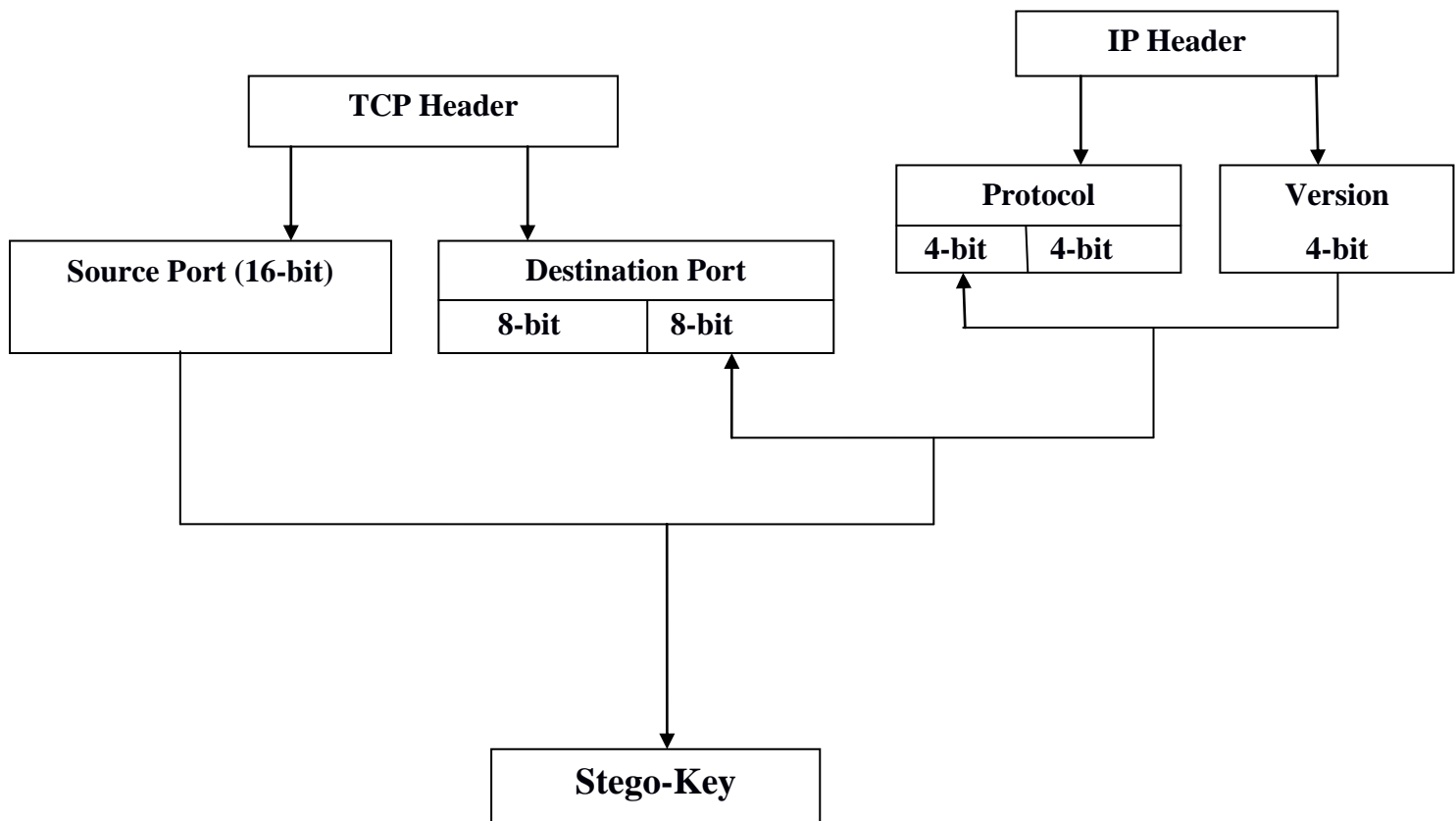


Fig.(3.3) Stego-key from IP and TCP Headers

Algorithm (3.5) Generate sequence number with srcport, destport, protocol and version

Input:

Data to be hidden, source port, destination port, protocol and version

Output:

Sequence number with hidden data

Begin

Step 1: converting data's characters to decimal

Step 2: converting result of step 1 to binary

Step 3: set result of step 2 to bindata variable

Step 4: converting source port to binary

Step 5: set result of step 4 to binsrcport variable

Step 6: converting destination port to binary

Step 7: set result of step 6 to bindestport variable

Step 8: converting protocol field value to binary

Step 9: set result of step 8 to binproto variable

Step 10: converting version field value to binary

Step 11: set result of step 10 to binver variable

Step 12: replacing second 4-bit of step 9's result with step 11's result

Step 13: set result of step 12 to binprotover variable

Step 14: replacing first 8-bit of step 7's result with step 13's result

Step 15: set result of step 14 to destprotover variable

Step 16: concatenation source and altered destination in their binary representation

Step 17: set result of step 16 to binsrcdestpv variable

Step 18: implementation XOR operation between result of step 3 and step 17

Step 19: set result of step 18 to binsteg variable
Step 20: converting binsteg value to decimal
Step 21: set result of step 20 to sequence number
Step 22: return sequence number
End

c. Create Packet

The Packet or datagram represents the final result in the Internet layer that consists of IP header that encapsulates TCP header and payload.

3.4.2 Send SYN packet

The generated packet with the hidden data was sent to the server to establish the connection. After Internet raw socket object was created and checked for creation error, the created packet and address was passed as parameters to sendto function. Then, it is sent.

3.4.3 Receive SYN-ACK packet

This part is concerned with the capturing SYN-ACK packet that was sent as a reply to SYN packet by server.

The destination IP address and destination port must be identified to begin receiving the replied packet. The analyzing fetched packet was completed after receiving it.

Through the analyzing process, IPHeader and TCPHeader were extracted. Unpack function of struct library in python was used for extraction process. Algorithm (3.6) is illustrated that.

Algorithm (3.6) Receive SYN-ACK packet

Input:

None

Output:

Received SYN-ACK packet

Begin:

Step 1: create an Internet raw socket object at layer 3

Step 2: check for creation error, if it is then system exit

Step 3: passing destination address and destination port to raw socket's binding function

Step 4: receive packet through raw socket's receiving function

Step 5: Analyze packet

Step 6: extract IP header and separate fields

Step 7: extract TCP header and separate fields

Step 8: return

End

3.5 Receiver Model (Packet Analysis)

As mentioned in chapter two, a packet analysis process helped in get live data and interpreted it. The intended recipient will get the hidden data by using packet analysis. Packet analysis is accomplished on the Internet layer and Ethernet for the two methods that are used to hide data.

In general, the capturing and analyzing packets processes for the two layers were different in addition to the part that is relating to receive the hidden data because the two methods are using different techniques for hiding, as it is illustrated in next section.

3.5.1 Capturing and analyzing process

Algorithm (3.7) is explaining capturing and analyzing at layer 3 (Internet layer) while algorithm (3.8) is explaining it at layer 2 (Ethernet). After that, a function that is using to get the hidden data will be explained later for the two embedding methods.

a. Capturing and Analyzing at Layer 3

This function represents a general function that is calling the necessary functions for capturing, analyzing and extraction the hidden data. Each of them will be explained later in separate section.

Algorithm (3.7) Packet Capturing and Analyzing at Layer 3

Input:

None

Output:

Hidden Data

Begin:

Step 1: calling capturing packet function

Step 2: calling analyzing IPHeader function

Step 3: calling analyzing TCPHeader function

Step 4: calling extraction function to get hidden data

Step 5: return

End

i. Capturing Packet

The purpose of this function is to capture SYN-ACK packet which is representing the replied packet by server.

Analyzing packet to extract IPHeader and TCPHeader also it will be explained in separate sections to get the concerned fields that is relating to hiding process. The extraction hidden data from the packet will also be explained in separate section. Algorithm (3.8) includes these steps.

Algorithm (3.8) Capturing Packet Function

Input:

None

Output:

Captured packet

Begin

Step 1: create an Internet raw socket object at layer 3

Step 2: check for creation error, if it is then system exit

Step 3: passing destination address and destination port to raw socket's binding function

Step 4: capturing packet through raw socket's receiving function

Step 5: return captured packet

End

ii. Analyzing IPHeader

The main role of this module is to extract fields of IPHeader from the packet. As mentioned earlier, the length of IPHeader is 20 – 60 bytes. Version and IPHeaderLength (IHL) each of them was 4-bit and combined in one byte. These two fields must be separated as follows :

$$\text{version} = \text{version_ihl} \gg 4$$

$$\text{ihl} = \text{version_ihl} \text{ and } F$$

Also, IP source and destination addresses must be converted to a host byte order since it was received as a network byte order. After defining total length as a global variable, the separation process for IP fields was done using unpack function.

iii. Analyzing TCPHeader

The role of this function is same as the previous one except that it is for TCPHeader not IP. As mentioned previously, the size of TCPHeader is 20- 60 bytes.

A note can be seen when TCPHeader is extracted. Source port will contain the value of destination host because the sender of SYN -ACK packet will be the server and the receiver will be the sender, and this applied for destination port too.

Another note is seen during the extraction of TCPHeader. Acknowledgement field that has (0) value in sending SYN packet, it will have value that is a summation of sequence number value + number of bytes that the server expected to receive after connection established.

The sequence number field has server's sequence number. Also, ACK bit field has set to 1. To implement analyzing process, the fields of source port, destination port, and acknowledgement defined as a global variables. Then, unpack function was used for separating fields.

b. Capturing and analyzing at layer 2

Capturing and analyzing process on Ethernet is different from the one implemented on layer 3 (Internet) because capturing on Ethernet will require dealing with device driver.

This function will capture packet after packet and analyze it by extracting the fields of headers at the same time until capturing a packet with the specified source port. As described in capturing on Internet layer, the source port value represents destination port of sender since the capturing process is on incoming packets from server.

After performing capturing and analyzing packets process, the data extraction function will be executed to get the hidden data as illustrated in algorithm (3.9).

Algorithm (3.9) Capturing and Analyzing at layer 2

Input:

Destination port

Output:

Hidden data

Begin

Step 1: reading destination port

Step 2: calling capturing and analyzing packet with destination port passing as

Input

Step 3: set source and destination ports, acknowledgement fields as global variables

Step 4: create packet raw socket object

Step 5: while condition true

Step 5.1: receiving packets through recvfrom function

Step 5.2: extract Ethernet header

Step 5.3: extract IP header

Step 5.4: extract TCP header

Step 5.5: check if destination port == the one passed as input then break,

Else go to step 5.1

Step 6: calling data extraction function (which is explained in next section)

Step 7: return

End

3.5.2 Data Extraction

As described in previous section of hiding data, two techniques were used to implement the hiding process.

In these techniques, the sequence number field is used for hiding but in extraction process, the acknowledgement field is used since it was resulting from sequence number value +1 with the other fields that were using through hiding process to get the hidden data as shown in Figure (3.4).

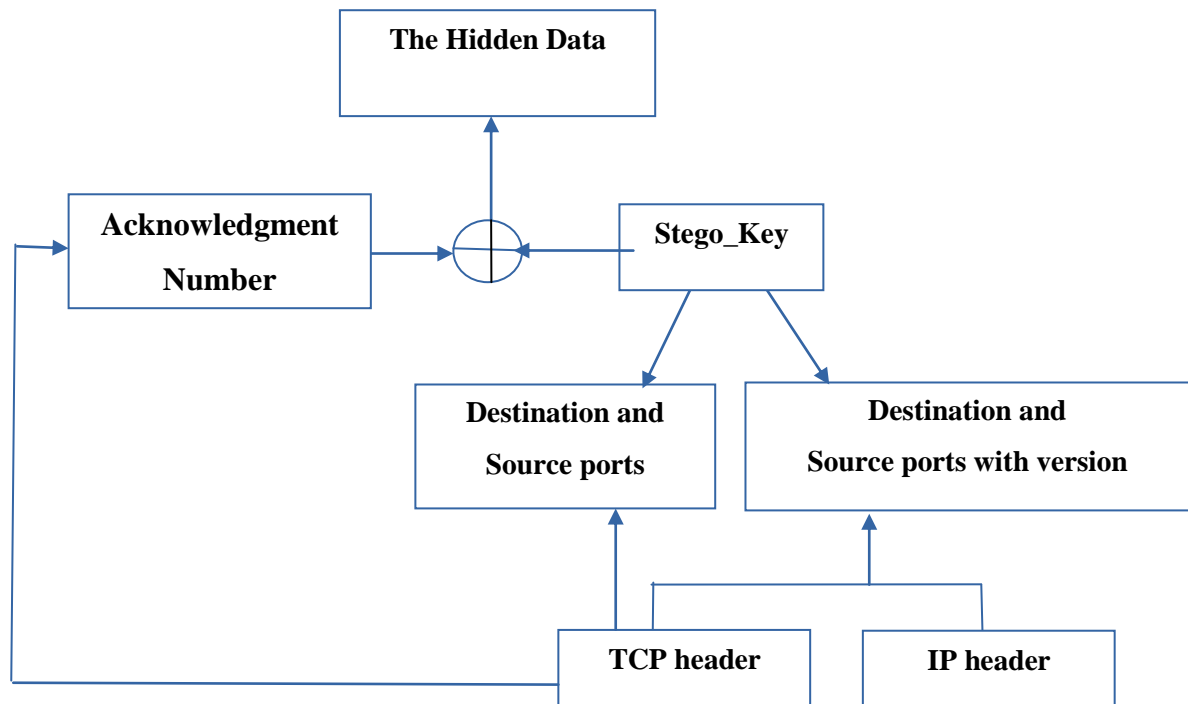


Fig.(3.4) Data Extraction Module

a. Data Extraction using TCP Header (Acknowledgement Number, Source Port and Destination Port)

The hidden data will be extracted from the acknowledgement field of SYN-ACK packet since the data is hidden in sequence number field of SYN packet. As mentioned previously, acknowledgment number value is sequence number value +1.

Attention is taken for source port and destination port because their values are replaced with each other with respect to sender and server. After that, the data is retrieved as illustrated in algorithm (3.10).

Algorithm (3.10) Data Extraction using acknowledgement number with srcdstports

Input:

Acknowledge number, source port, and destination port

Output:

Hidden Data

Begin

Step 1: converting acknowledgement number to binary

Step 2: set result to ackn variable

Step 3: converting source port to binary

Step 4: set result to binsrcport variable

Step 5: converting destination port to binary

Step 6: set result to bindestport variable

Step 7: concatenation source and destination ports in their binary representation

Step 8: set result to binsrctest variable

Step 9: implementation XOR operation between result of step 2 and step 8

Step 10: set result to binsteg variable

Step 11: converting result of step 10 to decimal

Step 12: set result to dessteg variable

Step 13: converting result of step 12 to data

Step 14: set result to hiddendata variable

Step 15: return hiddendata

End

b. Data Extraction using IP and TCP Header (Acknowledgement Number, Source Port, Destination Port, Protocol and Version)

The beginning of this function is the same as the previous one except additional fields is added. The difference is instead of combined source and destination ports field, source port (destination port in sender) field is altered through the processing that is described in section of generating sequence number with version and protocol fields as illustrated in algorithm (3.11).

Algorithm (3.11) Data Extraction with srcport, destport, protocol and version

Input:

Acknowledgement number source port, destination port, protocol and version

Output:

Hidden data

Begin

Step 1: converting acknowledgement number to binary

Step 2: set result to ackn variable

Step 3: converting source port to binary

Step 4: set result to binsrcport variable

Step 5: converting destination port to binary

Step 6: set result to bindestport variable

Step 7: converting protocol field value to binary

Step 8: set result to binproto variable

Step 9: converting version field value to binary

Step 10: set result to binver variable

Step 11: replacing second 4-bit of step 8 value with step 10 value

Step 12: set result to binprover variable

Step 13: replacing first 8-bit of step 6 value with step 12

Step 14: set result to destprover variable

Step 15: concatenation source and altered destination in their binary representation

Step 16: set result to binsrdestpv variable

Step 17: implementation XOR operation between result of step 2 and step 16

Step 18: set result to binsteg variable

Step 19: converting binsteg to decimal

Step 20: set result to dessteg variable

Step 21: converting result of step 20 to data

Step 22: set result to hiddendata variable

Step 23: return hiddendata

End

3.6 Server Model

The server model represents the tools that are described in chapter 2. These tools are netcat, netstat, and watch.

CHAPTER FOUR

CHAPTER FOUR

RESULTS

4.1 Introduction

The proposed system is achieved using Linux kernel raw sockets. Linux is a good environment provides the easiest access of raw sockets interface.

The task is performed with the aid of additional means. Wireshark is the first mean that is used to monitor traffic and ensure if a packet reached and viewing request and reply.

Virtualbox is the second tool that is using to setup ubuntu 15.10 for using netcat. Netcat represents the server that listens and waits for connection request from the client (sender of the hidden data).

Sending environment is supposed to be ideal (i.e. no loss packet, no retransmission occurred).

4.2 Results

The coding is executed through Python scripting. Transmission is sent from the host node to a predetermined IP address located at the beginning of the code. The address in this case was determined to be 192.168.3.102 or any other address set to virtual adapter. The sender should select an IP address with respect to the receiver's IP address to allow for delivery.

Whether changing iptables (see Appendix A) or not, the hidden data is sent. The execution of program for the two methods in embedding and extraction is shown below. The chosen word to be hidden was (help). After applying algorithm (3.2), the result will be as shown in Figure (4.1)


```
The Source Address is : 192.168.3.107
The Destination Address is : 192.168.3.102

The Data to be Hidden is : help
```

Fig. (4.1) source and destination addresses with the data- First method

Then, the TCP header was created after applying algorithm (3.3) as shown in Figure (4.2); the Figure displayed only the source and destination ports because they had the active role in hiding process.

```
The Source Port is : 27542
The Destination Port is : 8080
```

Fig (4.2) source and destination ports – First method

The sequence number with the hidden data was generated through creation of TCP header. By applying algorithm (3.4), the sequence number with the hidden data is created using the first method for hiding. As described in chapter 3, the first method includes source and destination ports as a stego-key. The Figure (4.3) explains that.

```
The Generation of Sequence Number is started .....
Message to binary = 01101000011001010110110001110000
Source port to binary = 0110100111010110
Destination port to binary = 0000100111111000
Concatination of source and destination ports in binary= 01101001110101100000100111111000
XOR of Message with Source and Destination ports = 00000001101100110110010110001000
The Generated Sequence Number is : 296144256
```

Fig. (4.3) Data hiding process – First method – First execution

As seen, the sequence number value is (296144256) represents the decimal result of XOR operation between the data to be hidden (help) and the stego-key.

To get the hidden data for this method, packet analysis on Internet layer or Ethernet is implemented. Figure (4.4) displays the result of algorithm (3.8) for capturing and analyzing at layer 3.

```
Capturing and Analyzing SYN-ACK Packet is begining ....
Source Address : 192.168.3.102
Destination Address : 192.168.3.107
Source Port : 8080
Destination Port : 27542
Sequence Number : 1023081466
Acknowledgement : 296144257
Data :
```

Fig. (4.4) Capturing and Analyzing – Layer 3- First execution

As seen, acknowledgement number value is (296144257) that represent the sequence number value +1. Algorithm (3.13) is applied to extract the hidden data for the first method of hiding process as shown in Figure (4.5).

```
Extraction of the Hidden Data is started ....
Acknowelgement to binary = 00000001101100110110010110001000
Destination port to binary = 0110100111010110
Source port to binary = 0000100111111000
Concatination of source and destination ports in binary= 01101001110101100000100111111000
XOR of Acknoweldegment with Source and Destination ports is 01101000011001010110110001110000

The Hidden Data is : help
```

Fig (4.5) Data extraction process – First method- First execution

When algorithm (3.2) is applied again, the result will be the same as in Figure (4.1) while the result of applying algorithm (3.3) is different because the source port is generated randomly as shown in Figure (4.6)

```
The Source Port is : 22666
The Destination Port is : 8080
```

Fig (4.6) source and destination ports – First method – Second execution

so, the sequence number that is generated with respect to these new values after applying algorithm (3.4) is different as it shown in Figure (4.7)

```
The Generation of Sequence Number is started .....
Message to binary = 01101000011001010110110001110000
Source port to binary = 0101000100011010
Destination port to binary = 0000100111111000
Concatination of source and destination ports in binary= 01010001000110100000100111111000
XOR of Message with Source and Destination ports = 0011100101111110110010110001000
The Generated Sequence Number is : 296156828
```

Fig (4.7) Data hiding process – First method – Second execution

The algorithm (3.12) and is applied for capturing and analyzing packet at Ethernet. Figure (4.8) shows the result of this algorithm and Figure (4.9) shows the result of algorithm (3.13) since the extraction process for the first method is the same while the capturing and analyzing processes are different

```
Enter port number to listen on: 8080
Capturing and Analyzing SYN-ACK Packet is begining ....
Source Port is : 22666
Destination port is : 8080
Acknowledgment Number is : 296156829
```

Fig (4.8) Capturing and Analyzing – Ethernet – First method- Second execution

As seen, the acknowledgement value is (296156829) that results from the sequence number value in Figure (4.7) +1.

```

Extraction of the Hidden Data is started ....
Acknowledgement to binary = 0011100101111110110010110001000
Destination port to binary = 0000100111111000
Source port to binary = 0101000100011010
Concatination of source and destination ports in binary= 01010001000110100000100111111000
XOR of Acknowledgement with Source and Destination ports 01101000011001010110110001110000

```

```
The Hidden Data is : help
```

Fig (4.9) Data extraction process – First method- Second execution

Although the data is the same as in Figure (4.1) but different sequence number is generated as in Figures (4.3), (4.7) respectively.

The second method for hiding data represents the result of applying algorithm (3.5) as it shown in Figure (4.12). As mentioned in chapter 3, the used stego_ key for hiding data is the combination of IP and TCP headers fields. But before that applying of algorithm (3.2) and (3.3) is necessary since the fields of stego_key is different as they are shown in Figures (4.10), (4.11) respectively.

```

The Source Address is : 192.168.3.107
The Destination Address is : 192.168.3.104
The version value is : 4
The protocol value is : 6

The Data to be hidden is : help

```

Fig (4.10) source and destination addresses, version, protocol, data- Second method

```

The Source Port is : 31696
The Destination Port is : 8080

```

Fig (4.11) source and destination ports- Second method- First execution

```

The Generation of Sequence Number is started ....
Message to binary = 01101000011001010110110001110000
Source port to binary = 00001011111011110
Destination port to binary = 0000100111111000
Protocol to binary = 01100000
Version to binary = 0010
Protocol with Version to binary = 01100010
Destination port with Protocol and Version to binary = 0110001011111000
Concatination of source port and destination port with Protocol and Version in binary=
000010111110111100110001011111000
XOR of Message with Source port and Destination port with Protocol and Version
01100011101110110000111010001000
The Generated Sequence Number is : 292609478

```

Fig (4.12) Data hiding process – Second method – First execution

Figure (4.13) displays the result of algorithm (3.8) for capturing and analyzing at layer 3 for the second method of hiding.

```

Capturing and Analyzing SYN-ACK Packet is begining ....
Source Address : 192.168.3.104
Destination Address : 192.168.3.107

Source Port : 8080
Destination Port : 31696
Acknowledgement : 292609479

```

Fig. (4.13) Capturing and Analyzing – Layer 3- Second method – First execution

Figure (4.14) displays the extraction process to get the hidden data for the second method

```

Extraction of the Hidden Data is started ....
Acknoweldgement to binary = 01100011101110110000111010001000
Destination port to binary = 00001011111011110
Source port to binary = 0000100111111000
Protocol to binary = 01100000
Version to binary = 0010
Protocol with Version in binary = 01100010
Source port with Protocol and Version in binary = 0110001011111000
Concatination of Destination port and Source port with Protocol and Version in binary=
000010111110111100110001011111000
XOR of Acknowledgment with Destination port and Source port with Protocol and Version =
01101000011001010110110001110000

The Hidden Data is : help

```

Fig. (4.14) Data extraction process – Second Method- First execution

For the second method of hiding and capturing and analyzing on Ethernet is used, the result of applying algorithm (3.2) will be the same as it in Figure (4.10) while the result of applying algorithm (3.3) is different because the source port is generated randomly as shown in Figure (4.15).

```
The Source Port is : 61652
The Destination Port is : 8080
```

Fig (4.15) source and destination ports- Second method- Second execution

After applying algorithm (3.5) again but for new execution, the result of sequence number will be different from the one that is shown in Figure (4.12) because source port is different since it is generated randomly for the second execution. Figure (4.16) explains that and shows (292574914) as the value of the generated sequence number.

```
The Generation of Sequence Number is started .....
Message to binary = 01101000011001010110110001110000
Source port to binary = 0010101100001111
Destination port to binary = 0000100111111000
Protocol to binary = 01100000
Version to binary = 0010
Protocol with Version to binary = 01100010
Destination port with Protocol and Version to binary = 0110001011111000
Concatination of source port and destination port with Protocol and Version in binary=
00101011000011110110001011111000
XOR of Message with Source port and Destination port with Protocol and Version
01000011011010100000111010001000
The Generated Sequence Number is : 292574914
```

Fig (4.16) Data hiding process – Second method – Second execution

Figure (4.17) displays the use of capturing and analyzing processes on Ethernet to get the hidden data for the second method of hiding.

```

Enter port number to listen on: 8080
Source Port is : 61652
Destination port is : 8080
Acknowledgment Number is : 292574915

```

Fig (4.17) Capturing and Analyzing – Ethernet – Second method- Second execution

It is clearly that the acknowledgement value is (292574915). The extraction process of the hidden data on this type of capturing is shown in Figure (4.18)

```

Extraction of the Hidden Data is started ....
Acknowledge to binary = 01000011011010100000111010001000
Destination port to binary = 0010101100001111
Source port to binary = 0000100111111000
Protocol to binary = 01100000
Version to binary = 0010
Protocol with Version in binary = 01100010
Source port with Protocol and Version in binary = 0110001011111000
Concatination of Destination port and Source port with Protocol and Version in binary=
00101011000011110110001011111000
XOR of Acknowledgment with Destination port and Source port with Protocol and Version
01101000011001010110110001110000

The Hidden Data is : help

```

Fig. (4.18) Data extraction process – Second Method- Second execution

The treatment of values in embedding and extraction processes includes converting all values to binary because the logical XOR that is supposed deals with binary values.

Capturing packet is proceeded in two methods as seen above. One of them represents direct method because it is binding to specific socket address (i.e. combination of IP destination address and port address) while the other represents indirect because it captured all incoming packets and specific condition causes stopping the capturing.

For implementing three way handshakes, iptables must be changed through the following statement executed through terminal as shown in Figure (4.19).


```

abeer@abeer-HP-15-Notebook-PC:~$ sudo iptables -A OUTPUT -p tcp --tcp-flags RST
RST -j DROP
[sudo] password for abeer:
abeer@abeer-HP-15-Notebook-PC:~$
    
```

Fig. (4.19) Changing iptables values

To ensure that all packets are transmitted, the packet sniffing program Wireshark will be used to collect all transmitted packets through my network. As shown in Figure (4.20), the SYN packet of the proposed system is sent from "192.168.0.104" source address to "192.168.0.105" destination address and "7365" random source port to "8080" destination port. The analyzed packet can be shown for the two protocols IP and TCP. In Figure (4.21), IPHeader of SYN packet is shown.

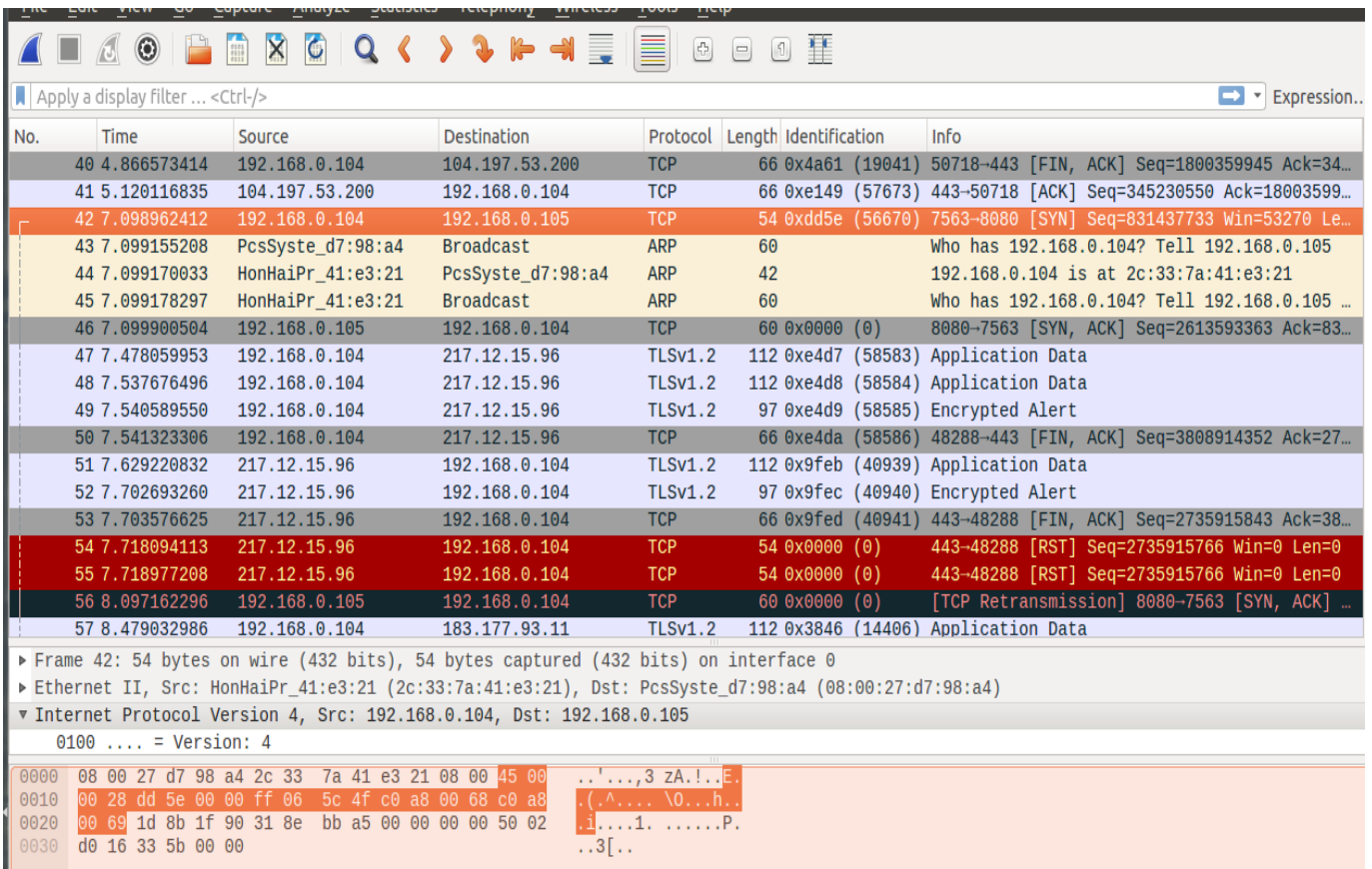


Fig (4.20) SYN packet


```

▶ Frame 42: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
▶ Ethernet II, Src: HonHaiPr_41:e3:21 (2c:33:7a:41:e3:21), Dst: PcsSyste_d7:98:a4 (08:00:27:d7:98:a4)
▶ Internet Protocol Version 4, Src: 192.168.0.104, Dst: 192.168.0.105
  0100 .... = Version: 4
▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 40
  Identification: 0xdd5e (56670)
▶ Flags: 0x00
Protocol: TCP (6)
Header checksum: 0x5c4f [validation disabled]
[Header checksum status: Unverified]
Source: 192.168.0.104
Destination: 192.168.0.105
[Source GeoIP: Unknown]
0000  08 00 27 d7 98 a4 2c 33 7a 41 e3 21 08 00 45 00  ..'...,3 zA.!...E.
0010  00 28 dd 5e 00 00 ff 06 5c 4f c0 a8 00 68 c0 a8  .(.^.... \0...h..
0020  00 69 1d 8b 1f 90 31 8e bb a5 00 00 00 00 50 02  .i....1. ....P.
0030  d0 16 33 5b 00 00  ..3[.

```

Fig. (4.21) IPHeader of SYN packet

and TCPHeader is seen in Figure (4.22). It clearly appears values of source and destination ports, sequence number with the hidden data, acknowledgement number value and SYN bit flag. A very important notation is seen that sequence number has value because the type of packet is SYN packet while acknowledgement number has

no value.

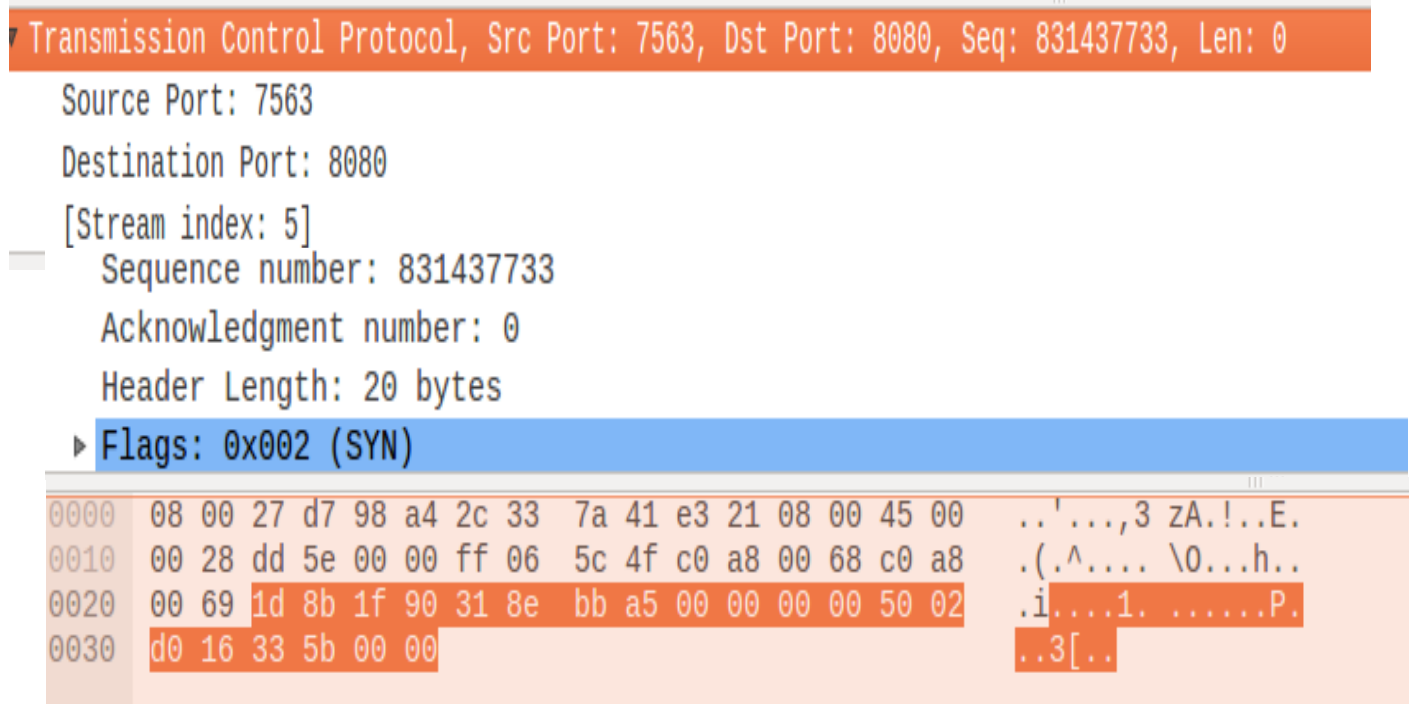


Fig (4.22) TCPHeader of SYN packet

The SYN-ACK packet can be seen as shown in Figure (4.23). The receiver of the hidden data captures this type of packet to extract data from it.

Time	Source	Destination	Protocol	Length	Identification	Info
35 4.760690478	217.12.15.96	192.168.0.104	TCP	54	0x0000 (0)	443-48298 [RST] Seq=4217307166 Win=0 Len=0
36 4.760707184	217.12.15.96	192.168.0.104	TCP	54	0x0000 (0)	443-48298 [RST] Seq=4217307166 Win=0 Len=0
37 4.865921346	104.197.53.200	192.168.0.104	TLSv1.2	119	0xe147 (57671)	Encrypted Alert
38 4.865945940	192.168.0.104	104.197.53.200	TCP	66	0x4a60 (19040)	50718-443 [ACK] Seq=1800359945 Ack=3452305...
39 4.866529423	104.197.53.200	192.168.0.104	TCP	66	0xe148 (57672)	443-50718 [FIN, ACK] Seq=345230549 Ack=180...
40 4.866573414	192.168.0.104	104.197.53.200	TCP	66	0x4a61 (19041)	50718-443 [FIN, ACK] Seq=1800359945 Ack=34...
41 5.120116835	104.197.53.200	192.168.0.104	TCP	66	0xe149 (57673)	443-50718 [ACK] Seq=345230550 Ack=18003599...
42 7.098962412	192.168.0.104	192.168.0.105	TCP	54	0xdd5e (56670)	7563-8080 [SYN] Seq=831437733 Win=53270 Le...
43 7.099155208	PcsSyste_d7:98:a4	Broadcast	ARP	60		Who has 192.168.0.104? Tell 192.168.0.105
44 7.099170033	HonHaiPr_41:e3:21	PcsSyste_d7:98:a4	ARP	42		192.168.0.104 is at 2c:33:7a:41:e3:21
45 7.099178297	HonHaiPr_41:e3:21	Broadcast	ARP	60		Who has 192.168.0.104? Tell 192.168.0.105
46 7.099900504	192.168.0.105	192.168.0.104	TCP	60	0x0000 (0)	8080-7563 [SYN, ACK] Seq=2613593363 Ack=83...
47 7.478059953	192.168.0.104	217.12.15.96	TLSv1.2	112	0xe4d7 (58583)	Application Data
48 7.537676496	192.168.0.104	217.12.15.96	TLSv1.2	112	0xe4d8 (58584)	Application Data
49 7.540589550	192.168.0.104	217.12.15.96	TLSv1.2	97	0xe4d9 (58585)	Encrypted Alert
50 7.541323306	192.168.0.104	217.12.15.96	TCP	66	0xe4da (58586)	48288-443 [FIN, ACK] Seq=3808914352 Ack=27...
51 7.629220832	217.12.15.96	192.168.0.104	TLSv1.2	112	0x9feb (40939)	Application Data
52 7.702693260	217.12.15.96	192.168.0.104	TLSv1.2	97	0x9fec (40940)	Encrypted Alert

Fig (4.23) SYN-ACK packet

SYN-ACK packet can be seen after analyzed both IPHeader and TCPHeader in Figures (4.24), (4.25) respectively.

```

▶ Frame 46: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▶ Ethernet II, Src: PcsSyste_d7:98:a4 (08:00:27:d7:98:a4), Dst: HonHaiPr_41:e3:21 (2c:33:7a:41:e3:21)
▼ Internet Protocol Version 4, Src: 192.168.0.105, Dst: 192.168.0.104
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 44
  Identification: 0x0000 (0)
  Protocol: TCP (6)
  Header checksum: 0xb8aa [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.0.105
  0000  2c 33 7a 41 e3 21 08 00 27 d7 98 a4 08 00 45 00  ,3zA.!... '.....E.
  0010  00 2c 00 00 40 00 40 06 b8 aa c0 a8 00 69 c0 a8  ,...@.@. ....i..
  0020  00 68 1f 90 1d 8b 9b c8 45 13 31 8e bb a6 60 12  .h..... E.1...`.
  0030  72 10 98 b8 00 00 02 04 05 b4 00 00              r.....
  
```

Fig (4.24) IPHeader of SYN-ACK packet

```

Sequence number: 2613593363
Acknowledgment number: 831437734
Header Length: 24 bytes
▶ Flags: 0x012 (SYN, ACK)
▼ Transmission Control Protocol, Src Port: 8080, Dst Port: 7563, Seq: 2613593363, Ack: 831437734, Len: 0
  Source Port: 8080
  Destination Port: 7563
  [Stream index: 5]
  0000  2c 33 7a 41 e3 21 08 00 27 d7 98 a4 08 00 45 00  ,3zA.!... '.....E.
  0010  00 2c 00 00 40 00 40 06 b8 aa c0 a8 00 69 c0 a8  ,...@.@. ....i..
  0020  00 68 1f 90 1d 8b 9b c8 45 13 31 8e bb a6 60 12  .h..... E.1...`.
  0030  72 10 98 b8 00 00 02 04 05 b4 00 00              r.....
  
```

Fig. (4.25) TCPHeader of SYN-ACK packet

The program is executed in two experiments. One of them is executed after changing Iptables. This means that the RST packet that operating system's kernel sends is dropping so that nc server is contentiously resends SYN-ACK packet until it receives ACK packet to establish connection or time-out and hence closing the connection as shown in Figure (4.26).

Time	Source	Destination	Protocol	Length	Identification	Info
40 4.866573414	192.168.0.104	104.197.53.200	TCP	66	0x4a61 (19041)	50718-443 [FIN, ACK] Seq=1800359945 Ack=34...
41 5.120116835	104.197.53.200	192.168.0.104	TCP	66	0xe149 (57673)	443-50718 [ACK] Seq=345230550 Ack=18003599...
42 7.098962412	192.168.0.104	192.168.0.105	TCP	54	0xdd5e (56670)	7563-8080 [SYN] Seq=831437733 Win=53270 Le...
43 7.099155208	PcsSyste_d7:98:a4	Broadcast	ARP	60		Who has 192.168.0.104? Tell 192.168.0.105
44 7.099170033	HonHaiPr_41:e3:21	PcsSyste_d7:98:a4	ARP	42		192.168.0.104 is at 2c:33:7a:41:e3:21
45 7.099178297	HonHaiPr_41:e3:21	Broadcast	ARP	60		Who has 192.168.0.104? Tell 192.168.0.105
46 7.099900504	192.168.0.105	192.168.0.104	TCP	60	0x0000 (0)	8080-7563 [SYN, ACK] Seq=2613593363 Ack=83...
47 7.478059953	192.168.0.104	217.12.15.96	TLSv1.2	112	0xe407 (58583)	Application Data
48 7.537676496	192.168.0.104	217.12.15.96	TLSv1.2	112	0xe4d8 (58584)	Application Data
49 7.540589550	192.168.0.104	217.12.15.96	TLSv1.2	97	0xe4d9 (58585)	Encrypted Alert
50 7.541323306	192.168.0.104	217.12.15.96	TCP	66	0xe4da (58586)	48288-443 [FIN, ACK] Seq=3808914352 Ack=27...
51 7.629220832	217.12.15.96	192.168.0.104	TLSv1.2	112	0x9feb (40939)	Application Data
52 7.702693260	217.12.15.96	192.168.0.104	TLSv1.2	97	0x9fec (40940)	Encrypted Alert
53 7.703576625	217.12.15.96	192.168.0.104	TCP	66	0x9fed (40941)	443-48288 [FIN, ACK] Seq=2735915843 Ack=38...
54 7.718094113	217.12.15.96	192.168.0.104	TCP	54	0x0000 (0)	443-48288 [RST] Seq=2735915766 Win=0 Len=0
55 7.718977208	217.12.15.96	192.168.0.104	TCP	54	0x0000 (0)	443-48288 [RST] Seq=2735915766 Win=0 Len=0
56 8.097162296	192.168.0.105	192.168.0.104	TCP	60	0x0000 (0)	[TCP Retransmission] 8080-7563 [SYN, ACK] ...
57 8.479032986	192.168.0.104	183.177.93.11	TLSv1.2	112	0x3840 (14400)	Application Data

Fig. (4.26) Execution of program after changing iptables

When iptables does not change, RST packet is sent by operating system's kernel normally and a connection is aborted as shown in Figure (4.27)

67	2.863697898	192.168.0.117	239.255.255.250	SSDP	167 0x594d (22861)	M-SEARCH * HTTP/1.1
68	2.893233046	HonHaiPr_41:e3:21	Broadcast	ARP	42	Who has 192.168.0.105? Tell 192.168.0.104
69	2.895102558	PcsSyste_d7:98:a4	HonHaiPr_41:e3:21	ARP	60	192.168.0.105 is at 08:00:27:d7:98:a4
70	2.895116569	192.168.0.104	192.168.0.105	TCP	54 0xed1c (60700)	4965-8080 [SYN] Seq=831436107 Win=53270 Le...
71	2.895328996	192.168.0.105	192.168.0.104	TCP	60 0x0000 (0)	8080-4965 [SYN, ACK] Seq=3843091365 Ack=83...
72	2.895350372	192.168.0.104	192.168.0.105	TCP	54 0xed1d (60701)	4965-8080 [RST] Seq=831436108 Win=0 Len=0
73	3.687744740	192.168.0.104	216.58.214.110	TCP	54 0xbef3 (48883)	44240-80 [ACK] Seq=2655087598 ACK=23285408...
74	3.805075890	216.58.214.110	192.168.0.104	TCP	54 0xc81 (7297)	[TCP ACKed unseen segment] 80-44240 [ACK] ...
75	7.282719216	192.168.0.104	50.31.164.174	TLSv1.2	119 0xb87b (47227)	Encrypted Alert
76	7.283380122	192.168.0.104	50.31.164.174	TCP	66 0xb87c (47228)	54752-443 [FIN, ACK] Seq=1534093104 Ack=19...
77	7.447782462	192.168.0.104	50.31.164.173	TCP	66 0x04c9 (1225)	41806-443 [ACK] Seq=2551372361 Ack=1900202...
78	7.680011600	50.31.164.174	192.168.0.104	TCP	66 0x98b0 (39088)	443-54752 [ACK] Seq=1913549468 Ack=1534093...
79	7.684122267	50.31.164.174	192.168.0.104	TCP	66 0x98b1 (39089)	443-54752 [FIN, ACK] Seq=1913549468 Ack=15...
80	7.684142361	192.168.0.104	50.31.164.174	TCP	66 0xb87d (47229)	54752-443 [ACK] Seq=1534093105 Ack=1913549...
81	7.689808870	50.31.164.174	192.168.0.104	TCP	66 0x98b3 (39091)	443-54752 [ACK] Seq=1913549469 Ack=1534093...
82	7.719520145	50.31.164.173	192.168.0.104	TCP	66 0x76e1 (30433)	[TCP ACKed unseen segment] 443-41806 [ACK]...
83	7.911490963	PcsSyste_d7:98:a4	HonHaiPr_41:e3:21	ARP	60	Who has 192.168.0.104? Tell 192.168.0.105
84	7.911505130	HonHaiPr_41:e3:21	PcsSyste_d7:98:a4	ARP	42	192.168.0.104 is at 2c:33:7a:41:e3:21

Fig. (4.27) Execution of program without changing iptables

TCPHeader of RST packet is shown in Figure (4.28). It is appeared that sequence number has value while acknowledgement number has no value. In addition, RST bit is set. This property is due to abort connection.


```
▼ Transmission Control Protocol, Src Port: 4965, Dst Port: 8080, Seq: 831436108, Len: 0
  Source Port: 4965
  Destination Port: 8080
  [Stream index: 5]
  Sequence number: 831436108
  Acknowledgment number: 0
  Header Length: 20 bytes
  ▶ Flags: 0x004 (RST)
```

Fig. (4.28) TCPHeader of RST packet

CHAPTER FIVE

CHAPTER FIVE

DISCUSSION, CONCLUSIONS, AND FUTURE WORK

5.1 Discussion

The environment of work was ubuntu 15.10. The work required dealing with TCP / IP protocol suite internally. Socket API represented interface for that. Different types of sockets existed depending on selected protocol of transport layer. In the beginning of implementation, TCP socket was used. Because this type of socket just sending data without creating headers, it had to be thinking about another type of socket enables the configuration of headers. Raw socket provided this ability. It deals with Internet layer directly.

Many of the existing tools based on the use of raw socket. Scapy (see Appendix B) one of these tools. After Scapy experienced, it proves that it is not suitable for performing task. A lot of problems appeared while using it. Such problems are how to convert configuration to deal with layer 3, send and receive from one port to another in loopback address ('127.0.0.1') because server concept requires listening for specific port specified to specific service.

The task is achieved in the use of pure raw socket for injection packet to traffic and capturing it from.

Server environment is needed. It was provided in using netcat utility in conjunction with netstat and watch. Surely, another copy of operating system is needed to operate server program. This is provided in set up virtual operating system inside virtualbox after install it.

Linux environment facilitated the work. The installation process of necessary packages is easy except it required fast Internet connection. Some of them installed

from Ubuntu Software Center while the other from availability on Internet as packages. For example Integrated Development Environment (IDE) for python programming language is provided from Ubuntu Software Center.

5.2 Conclusions

In this thesis, a steganography system based on TCP/IP protocols was constructed. The basic requirements for hiding data in such carrier were described.

A good combination for network programming was shown through python as a modern programming language and Linux. TCP/IP header fields were found as a good carrier for sensitive data to be hidid. With few minor operations with the field's contents, a secure carrier can be constructed. The serious issue was the restriction of field's sizes, since it is limited to a fixed size. This limitation may restrict the size of data in turn. Therefore, in order to send more data, the sender has to increase the connections.

As compared with [ROW97], the proposed methods are transferring four bytes through one connection while in [ROW97] one byte is transferred. The `stego_key` in [ROW97] is a static value summed with the ASCII of each character to be transferred while in the proposed methods its value is variable and contentiously changed. In both [CIO06] and [SIN13], two algorithms' for encrypting and compressing are applied to (4-byte) of data before transferring it. This leads to increase the used resources for processing.

5.3 Future work

1. The proposed system can be developed by adding interfaces to program. It can also make it as a tool running from terminal.

2. Other fields can be exploited for hiding as sequence number as like identification field in conjunction with sequence number. Other protocols can be used and its fields combined with fields of IP and TCP protocol headers.
3. Step 3 of three-way handshake can be implemented and sending normal data with hidden data in the header's fields as it happens in normal connection (i.e. without hiding data).
4. Packet analysis can be on incoming and outgoing packets in addition to the previous ones.
5. The data can be encrypted before hiding using modern algorithms of encryption like Advanced Encryption Standard (AES)

REFERENCES

References

- [AMI03] Amin, M. M., and et al, **“Information Hiding Using Steganography”**, Universiti Teknologi Malaysia, 2003.
- [BAU13] Baumgarten F., <http://net-tools.sourceforge.net/man/netstat.8.html>, 2013
- [BEA09] Beasley J. S., **“Networking, Second Edition”**, published by Pearson Education, 2nd Edition, 2009.
- [BIS16] Biswas R., Samir K. B., **“TCP Packet Steganography using SDA Algorithm”**, Journal of Scientific and Engineering Research, Vol 3, NO (2): p.p 47-51, 2016.
- [BUR07] Burns B. and et al, **“Security Power Tools”**, published by O’Reilly Media, Inc., 1st Edition, 2007
- [CIO11] Ciobanu R., and et al, **“Steganography and Cryptography Over Network Protocols”**, published in Roedunet International Conference (RoEduNet), 10th Edition, pp. 1-6. IEEE, 2011.
- [COL99] Coleman M. <https://linux.die.net/man/1/watch>, 1999
- [COM00] Comer D. E., **“Internetworking with TCP/IP: Principles, Protocols, and Architecture”**, published by Prentice Hall, Inc., Vol. 1, 4th Edition, 2000.

[DOR16] Dordal P. L., “**An Introduction to Computer Networks**”, free copy, released under [Attribution-NonCommercial-NoDerivs](#), Release 1.8.24, September 2016.

[FOR07] Forouzan B. A., “**Data Communications and Networking**”, published by McGraw-Hil Press, 4th Edition, 2007.

[FOR10] Forouzan B. A., “**TCP/IP Protocol Suite**”, published by McGraw-Hill Press, 4th Edition, 2010.

[FRY11] Frysigner M., <http://nc110.sourceforge.net/>, 2011

[GAN14] Gandhi, Dr. Ch. and et al “**Packet Sniffer – A Comparative Study**”, published in International Journal of Computer Networks and Communications Security, Vol. 2, No. 5, p.p. 179–187, May 2014.

[GUP14] Gupta R., Sunny G., and Anuradha S., “**Importance and Techniques of Information Hiding: A Review**”, International Journal of Computer Trends and Technology (IJCTT), Vol. 9, No. 5, March 2014.

[HOB] Hobbit, <https://linux.die.net/man/1/nc>,

[HUN02] Hunt C., “**TCP/IP Network Administration**”, Published by O’Reilly Media, Inc., 3rd Edition, April 2002.

[JAN10] Jankowski B., Mazurczyk W., and Szczypiorski, K., “**Information hiding using improper frame padding**”, published in proceedings of 14th international telecommunications networks strategy and planning symposium (NETWORKS), p.p.

[KAT00] Katzenbeisser S., Petitcolas F. A. P. “**Information Hiding Techniques for Steganography and Digital Watermarking**”, published by Artech House, 2000.

[KER07] Kerrisk M., <http://man7.org/linux/man-pages/man7/raw.7.html>, 2007

[KOZ05] Kozierok Ch. M., “**The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference**”, published by No Starch Press, Inc. , 2005.

[KUR13] Kurose J. F., Ross K. W., “**Computer Networking A top Down Approach**”, published by Pearson Education, Inc., 6th Edition, 2013.

[KUR14] Kurose J. F., Ross K. W., “**Computer Networking: A top Down Approach**” .ppt, <http://www-net.cs.umass.edu/kurose-ross-ppt-6e/>, 2014

[LEF86] Leffler S. J. and et al “**An Advanced 4.4BSD Interprocess Communication Tutorial**”, Computer Systems Research Group, Department of Electrical Engineering and Computer Science, University of California, Berkeley CA and Heterogeneous Systems Laboratory, Department of Computer Science, University of Maryland, College Park, 1986
<https://docs.freebsd.org/44doc/psd/21.ipc/paper.pdf>.

[MAR13] Marsic I., “**Computer Networks Performance and Quality of Service**”, published in Rutgers University, New Brunswick, New Jersey, 2013.

[MAZ08] Mazurczyk W., Szczypiorski, K., “**Steganography of VoIP Streams**”, In: Tari, Z. (ed.) On the Move to Meaningful Internet Systems (OTM 2008), Monterrey,

Mexico, LNCS, Vol. 5332, p.p 1001–1018, 9-14 November 2008.

[MAZ09] Mazurczyk W., Smolarczyk M., Szczypiorski K., **“Retransmission steganography and its detection”**, Springer-Verlag, 2009.

[MAZ13] Mazurczyk W., Smolarczyk M., Szczypiorski K., **“On Information Hiding in Retransmissions”**, published in telecommunication Systems: Modelling, Analysis, Design and Management, Vol. 52, issue 2, p.p.1113-1121, 2013.

[MIL14] Mileva A., Boris P., **“Covert Channels in TCP/IP Protocol Stack - extended version-”**, published in Central European Journal of Computer Science, Vol 4, NO (2). pp. 45-66, 2014.

[PER05] Perrone Prof. L. F. , **“CSCI 363 – Computer Networks”**, Department of Computer Science, Bucknell University, http://www.eg.bucknell.edu/~cs363/wordpress/?page_id=88 , 2005

[PET03] Peterson, L. L., Bruce, S. D., **“Computer Networks A Systems Approach”**, 3rd Edition, 2003.

[ROH11] Rohankar, N. D., Deorankar, A. V., Chatur, Dr. P. N., **“A Review of Literature on Design and Detection of Network Covert Channel”**, November 2011.

[ROW97] Rowland C. H., **“Covert Channels in the TCP/IP Protocol Suite”**, (First Monday Journal on the Internet), Vol. 2, No. (5), May, 1997.

[SAN11] Sanders C. **“Practical Packet Analysis”**, published by No Starch Press,

2nd Edition, 2011.

[SIN13] Singh J., Lalitsen Sh., “**Framework for Efficient Secure Steganographic Communication over Network Protocols**”, International Journal of Advanced Computer Research, Vol. 3, No. 4, Issue-13, p.p 146-150, December-2013.

[SZC03] Szczypiorski K., “**HICCUPS: Hidden Communication System for Corrupted Networks**”, In Proc. of: ACS’2003, Miedzyzdroje, Poland, p.p 31–40, October 22–24, 2003.

[TAN03] Tanenbaum, A. S., Wetherall, D. J., “**Computer Networks**”, published by Pearson Education, 4th Edition, 2003.

[TAN11] Tanenbaum, A. S., Wetherall, D. J., “**Computer Networks**”, published by Pearson Education, 5th Edition, 2011.

[WEL96] Welsh M., “**Linux Installation and Getting Started**”, published by Free Software Foundation, Inc., 1996

[ZAN07] Zander S., Armitage G., Brancha P., “**Survey of Covert Channels and Countermeasures in Computer Network Protocols**”, Vol. 9, No. 3, 3rd Quarter 2007.

APPENDIX A

iptables [HUN02]

For any operating system, firewall was existing. The main role of it was protecting network system from outside world by applying strict control for accessing. When destination of packet was host behind firewall, the delivery would be to firewall. In simple definition, firewall was a filtering router that detects undesirable traffic. Filtering router was a combination of routing capabilities of multi-homed Linux (i.e. does not forward packet to IP layer but process packet through application layer) and filtering features of iptables. The Linux kernel classified firewall traffic into three categories and different filter rules were applied to each of these categories

- **INPUT:** INPUT filter rules were tested for incoming traffic that was bound for a process on the local system before it was accepted.
- **OUTPUT:** OUTPUT filter rules were tested for outbound traffic that was initiated on the local system before it was sent.
- **FORWARD:** FORWARDING filter rules were tested for traffic from one external system that was bound for another external system.

When system was represented as a host, the INPUT and OUTPUT rules were used while the FORWARD rules were used when it was represented as a router. iptables also accepted user-defined additional to these categories. The set of rules defined by Linux kernel is shown in table (1) below.

Table (1)

Option	Function
-A	Appends rules to the end of a rule set.
-D	Deletes rules from a rule set.
-E	Renames a rule set.

-F	Removes all of the rules from a ruleset.
-I	Inserts a rule into a specific location in a rule set.
-L	Lists all rules in a ruleset
-N	Creates a user-defined rule set with the specified name.
-P	Sets the default policy for a chain
-R	Replaces a rule in a chain
-X	Deletes the specified user-defined rule set
-Z	Resets all packet and byte counters to zero.

Rules of firewall were a combination of filter with the packets that were matched and action taken if a packet and filter was matching. The action could either be a standard policy or jumping to rule set that was representing user-defined. The command line **-j target** was either a user-defined ruleset or a standard policy for packet handling. The name of a ruleset or standard policy that identified by a keyword was represented **target**. These keywords are as follows:

- ACCEPT : packet was allowed to passing firewall
- DROP: Discard the packet.
- QUEUE: packet should be passed up to user space for processing.
- RETURN: returning to ruleset that called this ruleset in a user-defined ruleset.

The filters that were constructed using iptables command using different command-line parameters which was as follows:

- -p protocol: specifies the protocol which was applied by rule. Protocol value could be tcp, udp or icmp as keyword
- -s address [/mask] : specifies the source address of packet which was applied by rule. Its value could be a host name, network name, or IP address.
- --sport [port [: port]]: specifies the source port of packets which was applied by rule. The format port: port was represented as a range of ports could be identified.

- -d address [/mask]: specifies the destination address of packet which was applied by rule. Its value could be a host name, network name, or IP address.
- --dport [port [: port]: specifies the destination port of packets which was applied by rule. All traffic was bounded to a specific port was filtered.
- --icmp-type type: specifies ICMP type which was applied by rule. Validated message type number or name was referred to type.
- -i name: specifies the name of the input network which was applied by rule. The packet was affected by this rule which was received on this interface
- -o name: specifies the name of the output network which was applied by rule.

The packet was affected by this rule which was sent out this interface.

- -f: related with second and subsequent of fragmented packet that the rule was referred.

APPENDIX B

SCAPY [BUR07]

1. Introduction

Scapy is a Python program written to manipulate network packets. It differs from most other tools because it is not a shell command program but comes in the shape of an interpreter. Actually, Scapy uses the Python interpreter evaluation loop to let you manipulate classes, instances, and functions.

Scapy comes with some new concepts and paradigms that make it a bit different from other tools in the domain of networking tools. With Scapy, packets are materialized in the shape of class instances. Creating a packet means instantiating an object, and manipulating a packet means changing attributes or calling methods of this instance object.

The basic building block of a packet is a layer, and a whole packet is built by stacking layers on top of one another. For example, a DNS packet captured on an Ethernet link will be seen as a DNS layer stacked over a UDP layer, stacked over an IP layer, stacked over an Ethernet layer. Because of this layering, using objects allows for an almost natural representation and code implementation. By implementing packets as objects, creating a packet from scratch is done in one line of code while it would have taken many lines in C, even with the best libraries. This allows for ease of use, and the user can implement and experiment with theoretical attacks much faster.

Moreover, the logic of sending packets, sniffing packets, matching a query and a reply, presenting couples, and tables is always the same and is provided by Scapy. A new tool can be designed in three steps:

1. Create your set of packets.
2. Call Scapy's logic to send them, gather the replies, parse them, match stimuli and answers.
3. Display the result.

Scapy dissociates the information harvesting phase and the result analysis. For example, you have to send a specific set of packets when you want to do a test, a port scan, or a traceroute. The packets you get back contain more information than just the simple result of the test and may be used for other purposes. Scapy returns the whole capture of the sent and received packets, matched by stimulus-response couples. You can then analyze it offline, as many times as you want, without probing again. This reduces the amount of network traffic and exposure to being noticed or flagging some IDS.

This raw result is decoded by Scapy and usually contains too much information for a human being to interpret anything right away. In order to make sense of the data, you will need to choose an initial view of interpretation where a meaning may become obvious.

The drawback of this is that it requires many more resources than only keeping what is useful for the current interpretation. However, it can save time and effort afterwards. Also, refining an interpretation without a new probe is more accurate because you can guarantee that the observed object did not change between probes. Always working on the same probe's data guarantees consistency of observations.

2. Working with Scapy

Scapy is not a traditional shell command-line application. When you run it, it will provide you with a textual environment to manipulate packets. Actually, it will run the Python interpreter and provide you many objects and functions that will enable you to manipulate packets.

Scapy runs in a Python interpreter; because of this, you can leverage the full functionality of Python. This means that you will be able to use Python commands, loops, and the whole language when dealing with packets.

3. Creating and Manipulating Packets with Scapy

A network packet is divided into layers, and each layer is represented by a Python instance. Thus manipulating a network packet is done by playing with instances' attributes and methods representing the different layers of the packet.

Creating a packet is done by creating instances, one for each layer, and stacking them together.

4. Scapy's Limitations

While Scapy has numerous features, it does come with some quirks that the user should be aware of. The first is that Scapy is not designed for fast throughput. It is written in Python, has many layers of abstraction, and it is not very fast. Do not expect a packet rate higher than 6 Mbs per second. Because of these layers of abstraction, and because of being written in Python, it may also require a lot of memory. When dealing with large amounts of packets, packet manipulation becomes uncomfortable after about 216 packets.

Scapy is stimulus-response-oriented. While you could do it, handling stream protocols may become painful. This is clearly an area of improvement. Yet, for the moment, it is possible to play with a datagram-oriented protocol over a stream socket managed by the kernel.

It easily designs something that sniffs, mangles, and sends. This is exactly what is needed for some attacks. But you will be disappointed in terms of performance or efficiency if you expect Scapy to do the job of a router. Do not confuse Scapy with a production mangling router that you could obtain with Netfilter.

المخلص

قد تتعرض عملية التبادل داخل شبكة المنطقة المحلية أو عبر شبكة الإنترنت إلى السرقة أو التغيير أو التدمير من قبل الشخص المؤذي الذي يمثل تهديدا حقيقيا لعملية النقل وأيضا للمعلومات خاصة إذا كانت هذه المعلومات حساسة ومهمة وينبغي أن يكون الوصول إليها من قبل الشخص المصرح به فقط. ولهذا يجب تأمين هذه البيانات مقابل هذه التهديدات. وقد اقترح العديد من الأفكار في إطار مفهوم الأمن لحماية البيانات من هذه التهديدات مثل إخفاء محتوى الرسالة المرسله التي كانت تسمى التشفير أو إخفاء وجود هذه الرسالة التي سميت ستيغانوغرافي.

اقترحت طريقتان لإخفاء هذه البيانات. أحدها استخدمت حقول منفذ المصدر ومنفذ الوجهة من رأس بروتوكول التحكم في الإرسال (TCP) كمفتاح ستيغو. والآخر استخدم مزيج حقول منفذ المصدر ومنفذ الوجهة من رأس بروتوكول التحكم في الإرسال (TCP) مع حقول الإصدار والبروتوكول من رأس بروتوكول الإنترنت (IP). العملية تلخصت من خلال تنفيذ أو حصرية (XOR) بين تلك البيانات المطلوب إخفاؤها مع مفتاح ستيغو. تم اختيار حقل رقم تسلسل من رأس بروتوكول التحكم في الإرسال (TCP) ليكون الناقل للبيانات المخفية. أربعة أحرف تم تضمينها في هذا الحقل وإرسالها في اتصال واحد.

تختلف الطرق المقترحة عن الطرق الموجودة . واحدة منها تم إرسال حرف واحد من خلال إتصال واحد بينما في الطرق المقترحة يتم إرسال أربعة أحرف .بالإضافة إلى هذا الاختلاف، فإن مفتاح الستيغو الذي تم استخدامه أيضاً يختلف . لأنه سيتم جمع قيمة ثابتة مع (ASCII) الحرف. بينما في الطرق المقترحة هو متغير وعملية الجمع لم تُستخدم بل تم استخدام XOR . على الرغم من أنه في الطرق الأخرى تم إرسال أربعة أحرف ولكنها استخدمت العديد من الموارد للتنفيذ لأن الأحرف قد تم ضغطها وتشفيرها.



جمهورية العراق
وزارة التعليم العالي والبحث العلمي
جامعة النهرين
كلية العلوم
قسم علوم الحاسوب

إخفاء النص في حقل رقم التسلسل لحزمة موائيق الإنترنت

رسالة
مقدمة إلى كلية العلوم في جامعة النهرين كجزء من متطلبات
نيل درجة الماجستير في علوم الحاسوب

من قبل
عبير عيسى عبد
(بكالوريوس علوم الحاسوب / كلية العلوم / جامعة النهرين، ٢٠٠٩)

إشراف
د.جمال محمد كاظم