CHAPTER FIVE
Conclusions and Suggestions

5.1 Conclusions

During the design, implementation and test phases of this research project, a lot of remarks have been issued; the following are some of them:

1. The proposed system concerned with monitoring the shared resource sharing, this requires continual monitoring of the ports that involved in the resource sharing processes. In this thesis it had been shown that ports 139 NetBIOS and 445, (which are part of the ports used in TCP/IP protocol suite) are indicated for accessing shared files.
2. Monitoring ports 139 NetBIOS and 445 requires that the process of climbing the stack of TCP/IP protocol suite begun from the internet layer, that is by capturing the network traffic in that layer (i.e. IP packets).
3. Not all the captured packets are relevant to filing access events, so a packet filtering system is needed to filter out the unnecessary packets. Packets filtering is applied on the captured packets to filter only those involved with ports 139 and 445. Checking the ports in this work is a sort of monitoring that based of signature type (specifically, the port signature).
4. Not all packets belong to ports 139 and 445 are needed in monitoring unit, only some of them are used to serve the monitoring process in the proposed system.
5. The number of relevant packets depends on the nature of the user access (i.e., if a user’s access to network resources increased, then the number of transmitted packets will increase also, and if the access type is copy

and deal with large files or folders, then the number of associated packets will increase).

6. Indexing a large number of folders and files takes a significant space in memory, and it needs few minutes in the initialization stage. Since, the monitoring unit concerns only with shared resources, so it is necessary to index only the shared folders and files to reduce the space and time requirements of the indexing process.

5.2 Suggestions for Future Work

In the following some suggestions for future work are given to enhance the proposed monitoring and make it more effective:

1. Develop the network monitoring system to be capable to make defense against threats directed to filing system, by using back up mechanism to retrieve the deleted or modified files.
2. Develop the network monitoring system to be capable to prevent the filing system from unauthorized users' accesses (for example imposing some controls on the attributes of system files and change their attributes to be read only).
3. Using some dynamic and evolutionary systems to improve the monitoring process, for example we can use neural network, because it is capable to be trained on user's actions or commands.
4. Enhance the files indexing system by decreasing the required time for indexing all folders.
5. Develop the network monitoring system to be capable to retrieve any report for any user at any time, by design and build an archive system.

CHAPTER FOUR
PERFORMANCE TEST RESULTS

4.1 Introduction

This chapter is dedicated to present the results of the conducted tests to evaluate the performance of the proposed monitoring unit. The tests have been performed to evaluate the performance of monitoring for all considered types of file/folder access (i.e., Read, Write, Copy, and Delete accesses). The measure used in these tests is the required time for detecting and assigning each access when the host storage media are loaded with different numbers of files and folders.

Also, in this chapter the results of testing the performance of administration unit's components (i.e., index filing system and rules editor) are presented.

4.2 The Test Results of Monitoring Unit

In this section, the number of captured packets per minutes are measured to show the strength of the implemented filtering mechanism.

When clients' machine turn on and start establishing connections with server, the monitoring unit should start capturing all the incoming to server, some test conducted for few minutes to be as examples, to show the ratio between the relevant packets to all transmitted packets . In Table (4.1), the field *"monitoring time"* means the run time of the established monitoring unit, the field *"Total packets number"* refers to the number of packets that captured during the monitoring time, while the field *"relevant packets number"* indicate for the number of packets passed through five modules' of the established monitoring unit, they are relevant to the proposed monitoring system.

As shown in Table (4.1), the monitoring time was taken around to 5 minutes, while the total number of all captured packets was noticed varies in a wide range packets because it depends on the number of logged on users using network resources.

Table (4.1) The relevant number of packets and the corresponding total number of transmitted packets

Monitored Time (minute)	*Total Number of Captured Packets(packet)*	*Relevant Packets Number (packet)*
4.83	3969	16
5.12	1348152	5
4.80	7267	35
5.11	3429	42

In Table (4.1), one can seen that the number of captured *relevant packets* doesn't depend on *total number of captured packets*, for example, when the total number of captured packets is 1348152 and the relevant packets are 5 (% 0.0004). In other example, the total number of captured packets is 3429 while the relevant packets are 42 (% 1.2). These small relative ratios of relevant packets refers to the importance of packet's filtering stage that applied on the captured packets, this reduce can led to stable performance of other system's modules.

4.2.1 Testing Versus Read and Write Accesses

Read access event occur when a client tries to access some shared folders or files. While the Write access even occur when a client tries to access one of the storage network resources to write on any file, even if write access is denied by the server machine.

When a client wants to open any file from shared resources, all folders exist in area defined by file’s path must be accessed by that user. The proposed monitoring unit refers to Read access when a client had opened these folders, while when he opened the file and tries to update, the proposed monitoring unit registers a write access event was occurred.

The established monitoring unit was implemented on two different machines. Table (4.2) shows the properties of these machines, while Table (4.3) shows the test results of Read and Writes accesses to network resources.

Table (4.2) The hardware features of the used two different machine

Machine Name	***Machine Features***
Computer 1	Intel (R), Celeron (R), 1.4 GHz processor, 480 MB RAM
Computer 2	Intel (R), Celeron (R), 1.73 GHz processor, 504 MB RAM

In Table (4.3) three keys are considered; type of file, size of file, and the time. *Time* field refers to required time in applies five modules’ monitoring unit on some packets for read and write access.

Table (4.3) The test results of READ & WRITE access

File Name	***File Type***	***File Size***	***Time in Computer 1 (milliseconds)***				***Time in Computer 2 (milliseconds)***			
			Read	***Write***	***Copy***	***Delete***	***Read***	***Write***	***Copy***	***Delete***
File1	HTML	2 KB	1	3	0*	0*	16	0*	0*	0*
File2	HTML	73KB	0*	16	0*	0*	3	0*	0*	0*
File3	PDF	421KB	1	0*	0*	0*	0*	0*	0*	0*

** zero (0) indicate the measured time is not significant (it is less than 0.5 millisecond)*

4.2.2 The Test Results of Copy Access

This section illustrates the evaluation of time that required to apply the five modules of monitoring unit on copy access case by such a client. As shown in Table (4.4), computer 1 and computer 2 are two different machines and their properties are listed in Table (4.2).

The monitoring unit doesn't need take a significant time if the copied folder is empty or has a small size, or contains small amount of subfolders and files. But if the number of copied subfolders and files is high or their sizes are large then the time required for applying copy action will increased.

Table (4.4) The results of tests conducted on COPY access

Folder Name	***Folder details (subfolders / files)***	***Folder Size***	***Time in Computer 1 (milliseconds)***				***Time in Computer 2 (milliseconds)***			
			Read	***Write***	***Copy***	***Delete***	***Read***	***Write***	***Copy***	***Delete***
Folder1	Empty	0	0*	0*	0*	0*	0*	0*	0*	0*
Folder2	0/1	1.5 MB	0*	0*	0*	0*	0*	0*	0*	0*
Folder3	1/111	105MB	0*	0*	16	0*	0*	0*	16	0*
Folder4	3200/439	294MB	0*	0*	15	0*	0*	0*	16	0*
Folder5	2/22	1.3GB	0*	0*	31	0*	0*	0*	15	0*
Folder6	1/48	1.7 GB	0*	0*	94	0*	0*	0*	16	0*

** zero (0) indicate the measured time is not significant (it is less than 0.5 millisecond)*

4.2.3 The Test Results of Delete Access

Table (4.5) shows the time test result for delete access, the properties of computer 1 and computer 2 are shows in Table (4.2).

In delete access, the five modules' of the monitoring unit needed 1-2 milliseconds when applied on computer 2, while they needed more time when they applied on computer 1.

Table (4.5) The test results of DELETE access

Folder Name	Detail Sub/ files	File Size	Time in Computer 1 (milliseconds)				Time in Computer 2 (milliseconds)			
			Read	Write	Copy	Delete	Read	Write	Copy	Delete
Folder1	Empty	0	0*	0*	0*	0*	0*	0*	0*	0*
Folder2	0/1	1.5 MB	0*	0*	0*	0*	0*	0*	0*	0*
Folder3	0/195	238MB	0*	0*	0*	16	0*	0*	0*	2
Folder4	100/790	MB	0*	0*	0*	2	0*	0*	0*	1
File 1	-	4.7 MB	0*	0*	0*	0*	0*	0*	0*	0*
File 2	-	56.8 MB	0*	0*	0*	0*	0*	0*	0*	0*

** zero (0) indicate the measured time is not significant (it is less than 0.5 millisecond)*

4.3 The Test Results of Administration Unit

In this section the results of the conducted tests to evaluate the performance of the components of administration unit's (i.e., filing indexing module and rules editor modules) are discussed.

The administration unit is installed and works on server machine. Therefore, in the test four different machines have been used as server machine, the properties of these machines are illustrated in Table (4.6).

Table (4.6) The hardware features of the used server machines

Machine Name	*Machine Features*
Computer 1	Intel (R), Celeron (R), 1.4 GHz processor, 480 MB RAM
Computer 2	Intel (R), Celeron (R), 1.73 GHz processor, 504 MB RAM
Computer 3	Intel (R), Pentium (R), 3.00 GHz processor, 504 MB RAM
Computer 4	Intel (R), Pentium (R), 3.00 GHz processor, 1.9 GB RAM

4.3.1 The Test Results of Files Indexing Module

The files indexing module was used in three ways: first, this module was applied for indexing all folders and files which are exist on the shared storage resources of the machine. Second, it was used for indexing folders only and ignores files. Third, this module was used for indexing the shared folders only and ignores other folders.

The file indexing module was applied on four different machines, their names and HW features are listed in Table (4.6). Table (4.7) shows the number of folders and files exit on the hard disk of each server machine. Also, this table illustrates the time taken by three runs: (1) indexing all folders and files, (2) indexing folders only, and, (3) indexing the shared folders only. The needed time to index folders is much less than the time needed to index folders and files, while the indexing of shared folders had elapsed the shortest time.

The volume of the overhead (i.e., indexing information) needed for indexing all folders was 16.3 MB, while for indexing all files was 81.5 MB.

Table (4.7) The test result of index filing module

Machine Name	*Index Types*	*Folders Number*	*Files Number*	*Time (minutes)*	*Size (MB)*
Computer 1	Folders & Files	6596	85817	3.25	97.8
Computer 1	Folders	6596	-	0.17	16.3
Computer 1	Shared Folders	267	-	0.020	16.3
Computer 2	Folders & Files	3553	58862	1.43	97.8
Computer 2	Folders	3553	-	0.49	16.3
Computer 2	Shared Folders	9	-	0.07	16.3
Computer 3	Folders & Files	4825	49587	2.08	97.8
Computer 3	Folders	4825	-	0.52	16.3
Computer 3	Shared Folders	10	-	0.051	16.3
Computer 4	Folders & Files	4125	50136	1.90	97.8
Computer 4	Folders	4125	-	0.18	16.3
Computer 4	Shared Folders	15	-	0.0002	16.3

4.3.2 The Test Results of the Rules Editor Module

Rules editor module consists of two main sub modules (i.e., add rules and modify rules). Table (4.8) shows the test results of the "add rules" sub module. This sub module was applied on the four server machine in two different ways, in the first way it was applied on all folders exist in the hard disk, while in the second test it was applied on the shared folders only. In the two cases, the rules editor modules (add rules, modify rules) had consumed 15.3 MB from size of server's hard disk to registered the required overhead information.

Table (4.8) The test results of add rules module

Machine Name	***Folders Types***	***Folders Number***	***Time (minutes)***
Computer 1	All folders	6596	3.49
Computer 1	Shared folders	267	0.016
Computer 2	All folders	3553	1.48
Computer 2	Shared folder	9	0.010
Computer 3	All folders	3561	1.26
Computer 3	Shared folder	10	0.018
Computer 4	All folders	4125	3.11
Computer 4	Shared folder	15	0.003

The clients always tries to access the shared documents, but the number of shared documents varies from client to another, for example the administrator may permit a client to access some folders, while he may permit all folders to other clients.

Table (4.9) shows the test results of the "modify rules" sub module. One can see that this sub module was applied on the four server machines in two different ways, like "add rules" sub module.

Table (4.9) The test results of “modify rules” module

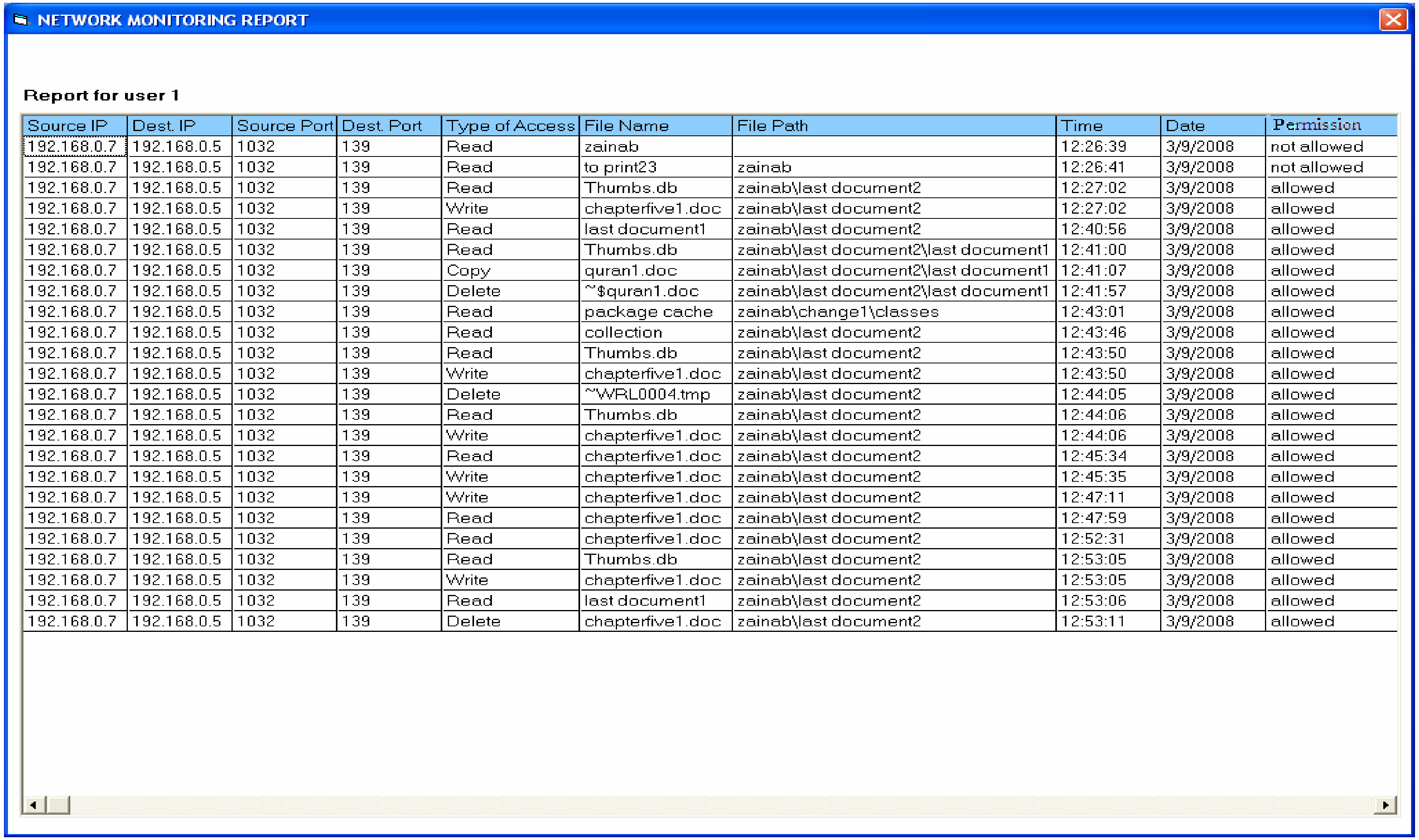
Machine Name	***Folders Types***	***Folders Number***	***Time (minutes)***
Computer 1	All folders	6596	2.662
Computer 1	Shared folders	267	0.014
Computer 2	All folders	3553	0.90
Computer 2	Shared folder	9	0.011
Computer 3	All folders	4825	0.92
Computer 3	Shared folder	10	0.026
Computer 4	All folders	4125	3.1
Computer 4	Shared folder	15	0.004

From test results listed in Tables (4.8) and (4.9), it can be noticed that the time required to scan the list of shared folders is much less than the time required scanning the list of all folders because the number of shared folders is less than the number of all folders. These results indicate the importance of applying the administration unit to manage the lists of shared modules and files exclusively, in order to minimize the overall management time taken by the administration unit.

4.4 Final Reports

Figure (4.1) shows the final report of user1. This report contains *source IP* and *destination IP* which are extracted from IP header, *source port* and *destination port* extracted from TCP header, *type of access* extracted from CIFS header, *file name* and *file path* extracted from NTCreate_andX and Deletedirectory headers. *Time* and *Date* fields represent the time and date of user access, from which the administrator know when users are access to the shared resources. The packet database for each client compared with rules database to decide which access is allowed and which is not. Finally the output illustrated in *Permission* field.

Figure (4.2) shows the report of user2, while Figure (4.3) shows the report of user3. Figure (4.4) shows the general report which contains the accesses of user1, user2 and user3.

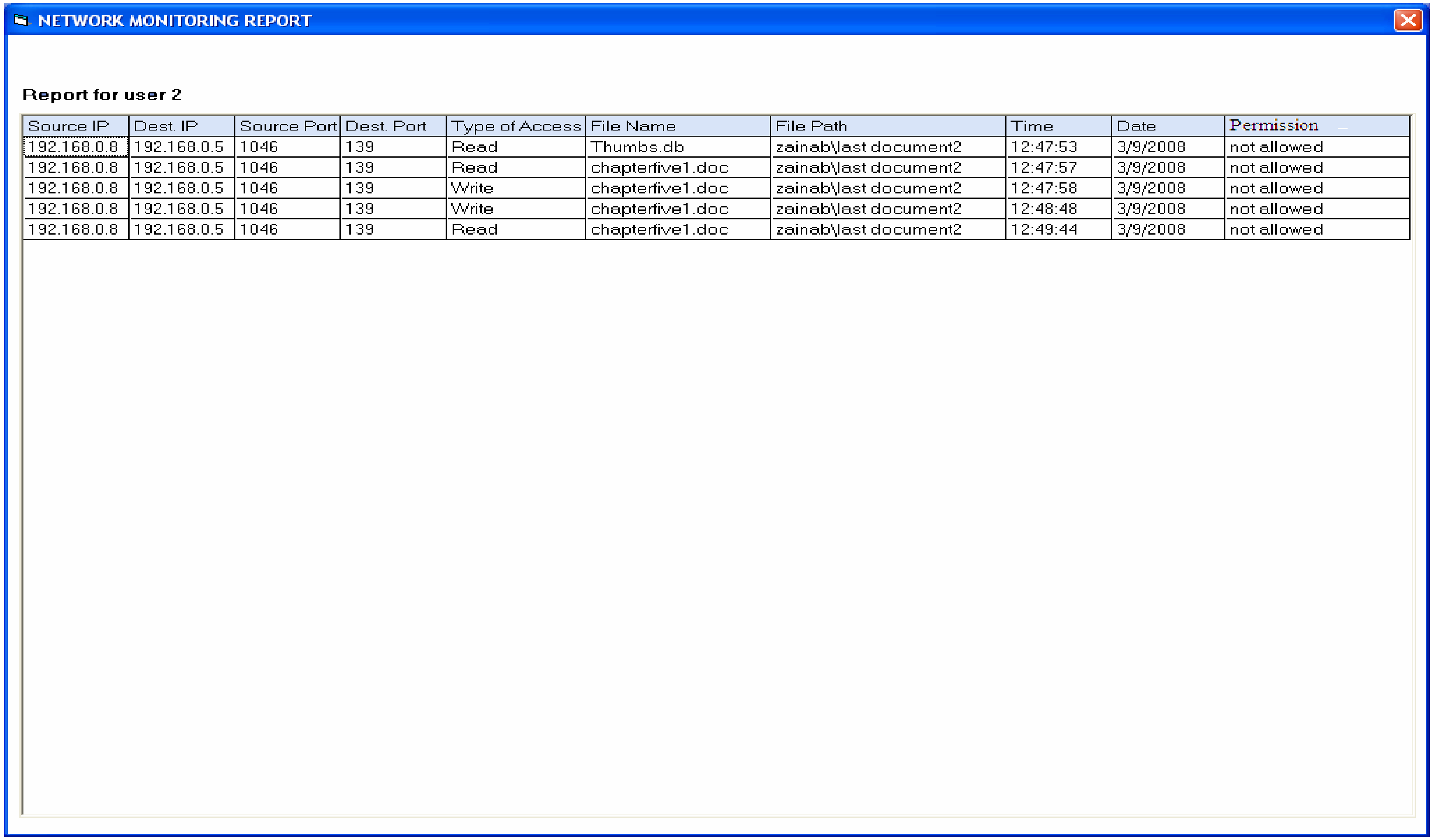


NETWORK MONITORING REPORT

Report for user 1

Source IP	Dest. IP	Source Port	Dest. Port	Type of Access	File Name	File Path	Time	Date	Permission
192.168.0.7	192.168.0.5	1032	139	Read	zainab		12:26:39	3/9/2008	not allowed
192.168.0.7	192.168.0.5	1032	139	Read	to print23	zainab	12:26:41	3/9/2008	not allowed
192.168.0.7	192.168.0.5	1032	139	Read	Thumbs.db	zainab\last document2	12:27:02	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Write	chapterfive1.doc	zainab\last document2	12:27:02	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	last document1	zainab\last document2	12:40:56	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	Thumbs.db	zainab\last document2\last document1	12:41:00	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Copy	quran1.doc	zainab\last document2\last document1	12:41:07	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Delete	~$quran1.doc	zainab\last document2\last document1	12:41:57	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	package cache	zainab\change1\classes	12:43:01	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	collection	zainab\last document2	12:43:46	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	Thumbs.db	zainab\last document2	12:43:50	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Write	chapterfive1.doc	zainab\last document2	12:43:50	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Delete	~WRL0004.tmp	zainab\last document2	12:44:05	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	Thumbs.db	zainab\last document2	12:44:06	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Write	chapterfive1.doc	zainab\last document2	12:44:06	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	chapterfive1.doc	zainab\last document2	12:45:34	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Write	chapterfive1.doc	zainab\last document2	12:45:35	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Write	chapterfive1.doc	zainab\last document2	12:47:11	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	chapterfive1.doc	zainab\last document2	12:47:59	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	chapterfive1.doc	zainab\last document2	12:52:31	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	Thumbs.db	zainab\last document2	12:53:05	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Write	chapterfive1.doc	zainab\last document2	12:53:05	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	last document1	zainab\last document2	12:53:06	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Delete	chapterfive1.doc	zainab\last document2	12:53:11	3/9/2008	allowed

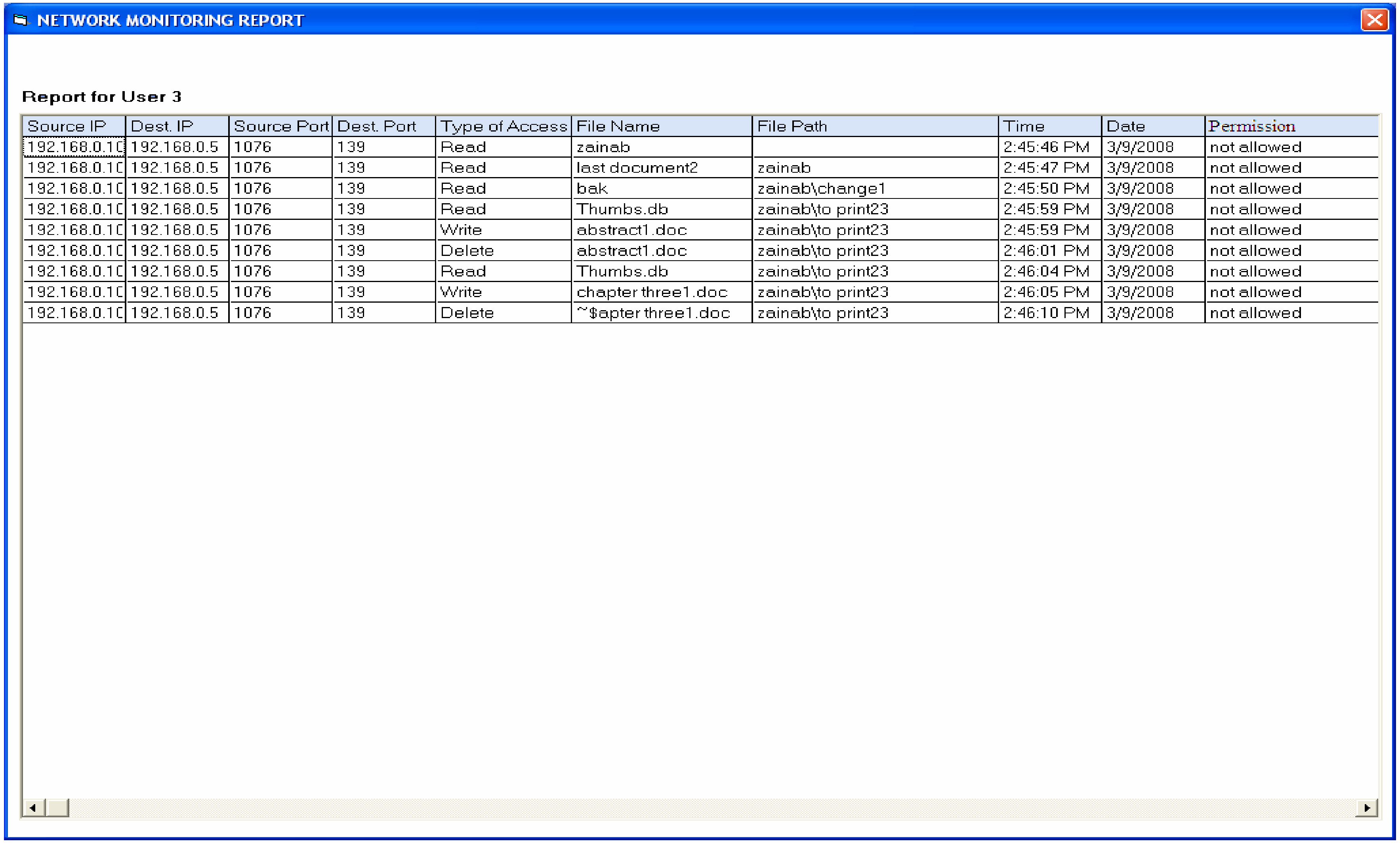
Fig. (4.1) A report for user1


NETWORK MONITORING REPORT

Report for user 2

Source IP	Dest. IP	Source Port	Dest. Port	Type of Access	File Name	File Path	Time	Date	Permission
192.168.0.8	192.168.0.5	1046	139	Read	Thumbs.db	zainab\last document2	12:47:53	3/9/2008	not allowed
192.168.0.8	192.168.0.5	1046	139	Read	chapterfive1.doc	zainab\last document2	12:47:57	3/9/2008	not allowed
192.168.0.8	192.168.0.5	1046	139	Write	chapterfive1.doc	zainab\last document2	12:47:58	3/9/2008	not allowed
192.168.0.8	192.168.0.5	1046	139	Write	chapterfive1.doc	zainab\last document2	12:48:48	3/9/2008	not allowed
192.168.0.8	192.168.0.5	1046	139	Read	chapterfive1.doc	zainab\last document2	12:49:44	3/9/2008	not allowed

Fig. (4.2) A report for user2

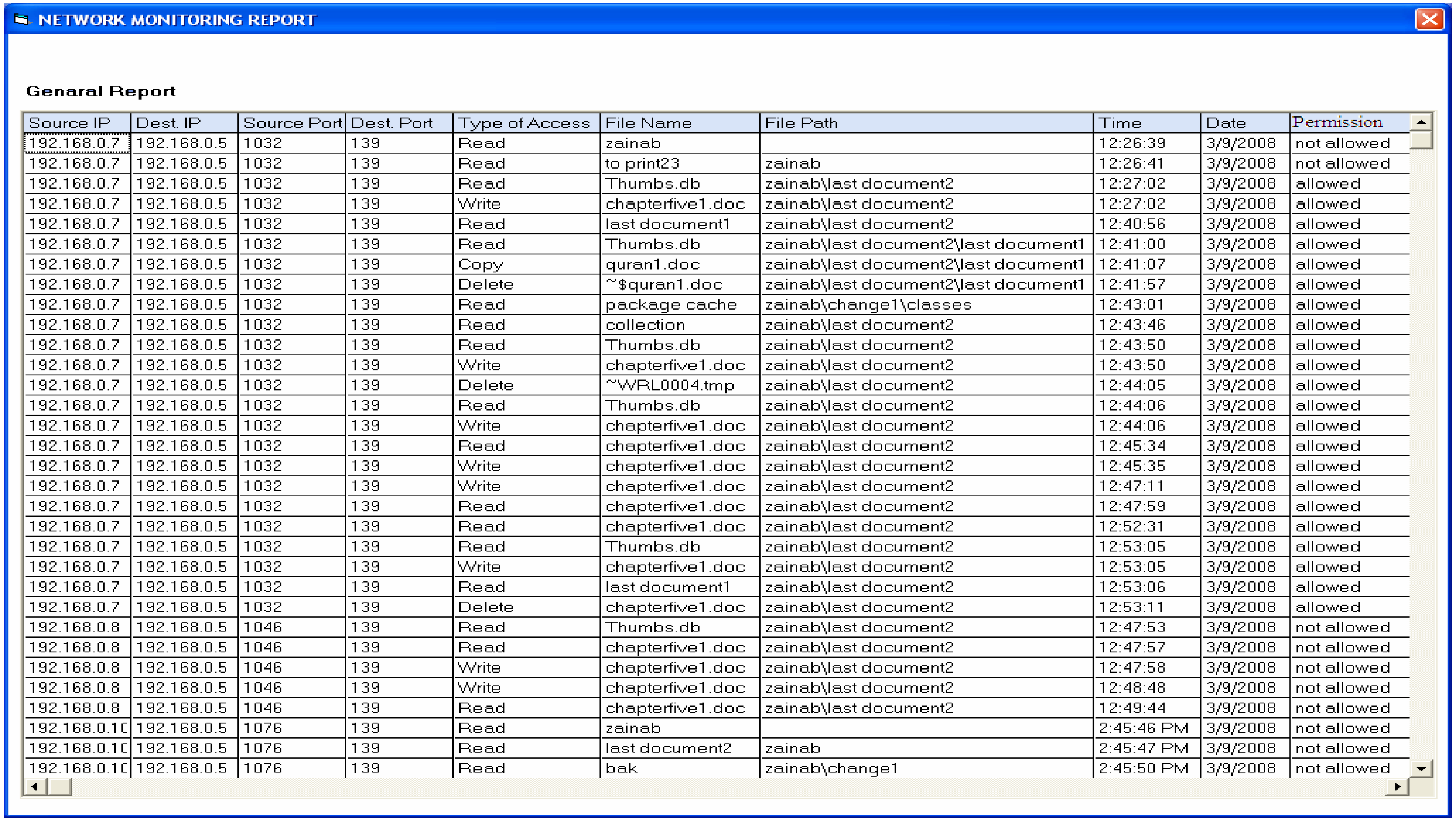


NETWORK MONITORING REPORT

Report for User 3

Source IP	Dest. IP	Source Port	Dest. Port	Type of Access	File Name	File Path	Time	Date	Permission
192.168.0.10	192.168.0.5	1076	139	Read	zainab		2:45:46 PM	3/9/2008	not allowed
192.168.0.10	192.168.0.5	1076	139	Read	last document2	zainab	2:45:47 PM	3/9/2008	not allowed
192.168.0.10	192.168.0.5	1076	139	Read	bak	zainab\change1	2:45:50 PM	3/9/2008	not allowed
192.168.0.10	192.168.0.5	1076	139	Read	Thumbs.db	zainab\to print23	2:45:59 PM	3/9/2008	not allowed
192.168.0.10	192.168.0.5	1076	139	Write	abstract1.doc	zainab\to print23	2:45:59 PM	3/9/2008	not allowed
192.168.0.10	192.168.0.5	1076	139	Delete	abstract1.doc	zainab\to print23	2:46:01 PM	3/9/2008	not allowed
192.168.0.10	192.168.0.5	1076	139	Read	Thumbs.db	zainab\to print23	2:46:04 PM	3/9/2008	not allowed
192.168.0.10	192.168.0.5	1076	139	Write	chapter three1.doc	zainab\to print23	2:46:05 PM	3/9/2008	not allowed
192.168.0.10	192.168.0.5	1076	139	Delete	~$apter three1.doc	zainab\to print23	2:46:10 PM	3/9/2008	not allowed

Fig. (4.3) A report for user3



NETWORK MONITORING REPORT

Genaral Report

Source IP	Dest. IP	Source Port	Dest. Port	Type of Access	File Name	File Path	Time	Date	Permission
192.168.0.7	192.168.0.5	1032	139	Read	zainab		12:26:39	3/9/2008	not allowed
192.168.0.7	192.168.0.5	1032	139	Read	to print23	zainab	12:26:41	3/9/2008	not allowed
192.168.0.7	192.168.0.5	1032	139	Read	Thumbs.db	zainab\last document2	12:27:02	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Write	chapterfive1.doc	zainab\last document2	12:27:02	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	last document1	zainab\last document2	12:40:56	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	Thumbs.db	zainab\last document2\last document1	12:41:00	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Copy	quran1.doc	zainab\last document2\last document1	12:41:07	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Delete	~$quran1.doc	zainab\last document2\last document1	12:41:57	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	package cache	zainab\change1\classes	12:43:01	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	collection	zainab\last document2	12:43:46	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	Thumbs.db	zainab\last document2	12:43:50	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Write	chapterfive1.doc	zainab\last document2	12:43:50	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Delete	~WRL0004.tmp	zainab\last document2	12:44:05	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	Thumbs.db	zainab\last document2	12:44:06	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Write	chapterfive1.doc	zainab\last document2	12:44:06	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	chapterfive1.doc	zainab\last document2	12:45:34	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Write	chapterfive1.doc	zainab\last document2	12:45:35	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Write	chapterfive1.doc	zainab\last document2	12:47:11	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	chapterfive1.doc	zainab\last document2	12:47:59	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	chapterfive1.doc	zainab\last document2	12:52:31	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	Thumbs.db	zainab\last document2	12:53:05	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Write	chapterfive1.doc	zainab\last document2	12:53:05	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Read	last document1	zainab\last document2	12:53:06	3/9/2008	allowed
192.168.0.7	192.168.0.5	1032	139	Delete	chapterfive1.doc	zainab\last document2	12:53:11	3/9/2008	allowed
192.168.0.8	192.168.0.5	1046	139	Read	Thumbs.db	zainab\last document2	12:47:53	3/9/2008	not allowed
192.168.0.8	192.168.0.5	1046	139	Read	chapterfive1.doc	zainab\last document2	12:47:57	3/9/2008	not allowed
192.168.0.8	192.168.0.5	1046	139	Write	chapterfive1.doc	zainab\last document2	12:47:58	3/9/2008	not allowed
192.168.0.8	192.168.0.5	1046	139	Write	chapterfive1.doc	zainab\last document2	12:48:48	3/9/2008	not allowed
192.168.0.8	192.168.0.5	1046	139	Read	chapterfive1.doc	zainab\last document2	12:49:44	3/9/2008	not allowed
192.168.0.1	192.168.0.5	1076	139	Read	zainab		2:45:46 PM	3/9/2008	not allowed
192.168.0.1	192.168.0.5	1076	139	Read	last document2	zainab	2:45:47 PM	3/9/2008	not allowed
192.168.0.1	192.168.0.5	1076	139	Read	bak	zainab\change1	2:45:50 PM	3/9/2008	not allowed

Fig. (4.4) The general report

CHAPTER ONE
GENERAL INTRODUCTION

1.1 Introduction

The world of computers has changed dramatically over the past three decades. Before twenty-five years ago, most computers were centralized and managed by data centers, computers were kept in locked rooms and staffs of people made sure they were carefully managed and physically secured. Links outside a site were unusual. Computer security threats were rare, and they were basically concerned with insiders: authorized users, accounts misuse, theft and vandalism, and so forth. These threats were well understood and dealt with using standard prevention techniques (like, computers behind locked doors, and accounting for all resources). While computing in the 1990's, and after, is radically different. Many systems are in private offices and labs, and they often managed by individuals or persons employed outside a computer center. Many systems are connected into the Internet, and from there around the world.

Security threats are different today. The advisers say "don't write your password down and put it in your desk" someone may find it. With world-wide Internet connections, someone could get into such system from the other side of the world and steal the password in the middle of the night when the building is locked up. Viruses and worms can be passed from machine to machine. The Internet allows the electronic equivalent of the thief who looks for open windows and doors, now a person can check hundreds of machines for vulnerabilities in a few hours.

System administrators and decision makers have to understand the security threats that exist, what the risk and cost of a problem would be, and what kind of action they want to take (if any) to prevent and respond to security threats. [RFC1244]

From a day-to-day perspective, network administrators and operators want assurance that the equipment they have deployed is behaving properly. They wish to know when the use of the network changes, requiring new hardware, or configuration changes on existing hardware. Although this information may be available from the hardware products themselves, network monitoring software provides flexibility to the network administrator and the opportunity to verify that the hardware is reporting properly. Administrators and operators also want to ensure that their networks are not being used for nefarious purposes. Network monitoring provides a window onto network use, possibly revealing malicious outsiders who have exploited one or more hosts on the network, or authorized insiders using the network for unauthorized purposes. [HUG06]

At (1990) [SHE07] a network monitoring system called “Network System Monitor (NSM) “ was developed in University of California. NSM was established to run on a SUN UNIX workstations. It was the first foray into monitoring network traffic, which uses traffic data as the primary data source. Before this time, most intrusion detection systems utilize the information extracted from operating system audit trails or keystroke monitors [HER90].

Intrusion Detection Systems (IDS) are an important component to protect computer systems and networks from abuse. Although intrusion detection technology is immature and should not be considered as a complete defense, it can play a significant role in overall security architecture. When an IDS is properly deployed, it can provide warnings indicating that a system is under attack, even if the system is not vulnerable to the specific attack. These warnings can help users alter their installation’s defensive posture to increase resistance to attack. In addition, an IDS can serve to confirm secure configuration and operation of other security mechanisms such as firewalls [MCH00].

1.2 Related Work

- **Molina 2001 [MOL01]:** He had implemented an auditory intrusion detection system (IDS) which is based on an out-of band approach (i.e., using integrity checking system), inside the auditor other IDS can be implemented. The auditor could sniff the traffic passing by the Ethernet card and alert the machine when detects some possible network attacks. To ensure and check the integrity of the file system, two approaches have been followed; the first approach is to create a secure database, which is usually composed of hashes of the important files, the second approach is to create digital signatures of sensitive data (such as executable files) using public keys.

- **Ali 2003 [ALI03]:** He had built an intrusion detection system using full packet monitoring technique to detect both known and new attacks. The most fundamental elements that need to be examined during traffic monitoring and analysis are source and destination IP address, TCP and UDP traffic, ICMP traffic.

- **Mustafa 2003 [MUS03]:** He had designed a system to detect intrusions caused by intrudes in network based system, the proposed system captures packet in the network and analyze them to check if they are normal or abnormal. The analysis combines anomaly and misuse Intrusion Detection to detect more kinds of intrusion. Anomaly detection was done by using back propagation neural network to learn the normal network traffic in order to detect the abnormal traffic. Misuse detection matches the current traffic with several attack signatures.

- **Viipuri 2004 [VII04]:** He had introduced a packet monitoring system with free packet capture and analysis software. Network traffic was analyzed in a core network consisting of multiple entry and exit points, which have been called points of presence (PoP). Each PoP was fitted with a traffic monitoring device listening on all incoming and outgoing packets to and from the network.

- **Pande 2005 [PAN05]:** He had introduced a network monitoring tool, called PickPacket, that handles the conflicting issues of network monitoring and privacy through its judicious use. PickPacket has four components: (1) PickPacket configuration file generator for assisting the user in setting up the parameters for capturing packets, (2) PickPacket packet filter for capturing packets, (3) PickPacket post-processor for analyzing packets, and (4) PickPacket data viewer for showing the captured data to the user.

- **Abed 2006 [ABE06]:** He had established a system called *"Low Level Security"* aim to protect NetBIOS by using firewall to deny of all external incoming requests from unauthorized users that try to access ports (137 UDP, 138 UDP, 139 TCP) through LAN, such requests may try to see or damage the private information and at the same time they may allow the passage of other incoming requests from authorized user. The proposed system uses packets filtering and firewall technology. The objective of this proposed NetBIOS firewall system is to protect the shared resources and information from unauthorized user who wants to see or damages the private information or share information. The system apply a fixed set of rules on the incoming packet to determine whether they will be allowed to pass or not, these rules are based on comparing the IP address and port number of the source and destination with those authorized IP, and port numbers listed in a table.

- **Hughes 2006 [HUG06]:** He focused on parsing application streams using standard parsing methods. Also, he proposed a machine-readable grammar for specifying stream protocols, and developed a parser generator to build parsers for network monitors. The proposed grammar is compatible with the grammar used to describe the existing protocols. In order to convert the existing specifications to the proposed format the researcher suggested that a network analyzer should climb the protocol stack from the physical and transport layers to the application layer.

1.3 Aim of Thesis

The aim of this research is to design and build a network security monitoring system which monitors all login users in a local area network and make security evaluation to their access to the network filing system. Also it is concerned with building a dedicated file indexing system which is required to check the users' authentication. Users' authentication is one of the more common security mechanisms; it requires defining the allowed privileges for each user, by which the user can access the network storage media. These rules are specified by the administrator which has the privilege to manage the whole system, and receives reports about any violation issued by one or more of the users. From those reports the administrator can recognize and assess the intrusion level.

1.4 Thesis Layout

The remaining part of this thesis consists of four chapters:

- **Chapter Two entitle “Network Protocols and Security“**

  This chapter is concerned with the definition of network models, Transport Control Protocol/ Internet Protocol (TCP/IP) suite, and its important protocols, specifically Server Message Block (SMB) protocol and Common Internet File System (CIFS) protocol. Also, some related topics in security and network security are defined. At the end of this chapter, the filing systems (its types, its access controls) are given.

- **Chapter Three entitle “Network Security Monitoring System Design ”**

  This chapter introduces the design and implementation steps of the proposed network monitoring system. Also in this chapter the established indexing system for the shared files is presented.

- **Chapter Four entitle “Performance Test Results “**

  This chapter introduces the results of tests to evaluate the performance of the proposed network monitoring system.

- **Chapter Five entitle “Conclusions and Suggestions”**

  This chapter introduces the derived conclusions and suggestions for future works are given.

CHAPTER TWO
NETWORK PROTOCOLS AND SECURITY

2.1 Network Fundamentals

A computer network is an interconnected system of computing devices that provides shared economical access to computer services. Networks are important since they provide several benefits (such as resource sharing, saving money, etc).

Networks can be divided into Local Area Networks (LANs), Metropolitan Area Networks (MANs), and Wide Area Network (WANs). Each network type has its own characteristic, technologies, and speeds. LANs cover a building and operate at high speeds. MANs cover a city (for example, the cable television system). WANs cover country or continent. LANs and MANs are unswitched (i.e., do not have routers); WANs are switched. Networks can be interconnected to form internetworks [TAN03].

Communication between computers in a network can be accomplished in two important fashions: (1) a broadcast fashion where systems transmit in general to the media accessible to all other systems, Figure (2.1) shows the two most common topologies found in a broadcast networks, and (2) a point-to–point fashion where each system transmits to

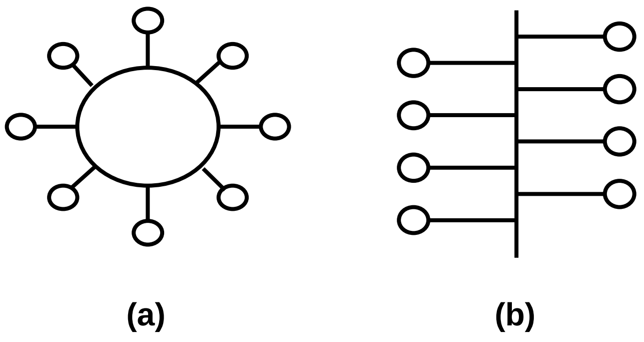


Fig. (2.1) Broadcast network topologies: (a) Ring, (b) Bus.

another specific system, Figure (2.2) depicts two other topologies found in point-to-point networks [FIS00].

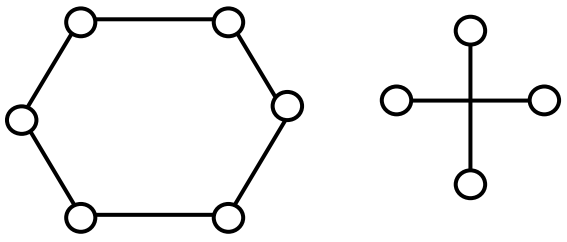


Fig. (2.2) Point-to-Point topologies: (a) Ring, (b) Star.

2.2 Network Models

A Network model is a protocol suite reflects the design or architecture to accomplished communication between different systems. Any network model usually consists of layers. Each layer of the model represents specific functionality [Hel00].

There are two important network models: (1) International Standard Organization/ Open System Interconnection (ISO/OSI) model and (2) Transmission Control Protocol/Internet Protocol (TCP/IP) model. The Protocols associated with ISO/OSI model are rarely used any more while the protocols of the TCP/IP model are widely used [TAN03]. TCP/IP works in a very similar manner to OSI model in that it takes a layered approach to provide network services [Hel00].

The OSI has 7-layers, each layer is build upon the layer below it to standardize and simplify communication between them. Conceptually, each layer is designed to provide specific services to the layer above it. This effectively hides the details of communication between lower level layers and the upper levels, and it serves to modularize the protocols for each layer. Also the purpose is using layers to create an environment where each layer on a system communicates with the same layer on another system by using the protocol that developed for that layer. Each subsequent lower level takes the information passed to it from the level above, and formats it to meet its protocol needs before sending it along; Figure (2.3) illustrates this process. In this figure, a seven layer model is assumed. Data to be sent is created at layer 7 of the source host and send to layer 7 of the destination host. The message only needs to be formatted in layer 7, and is then passed to the next level.

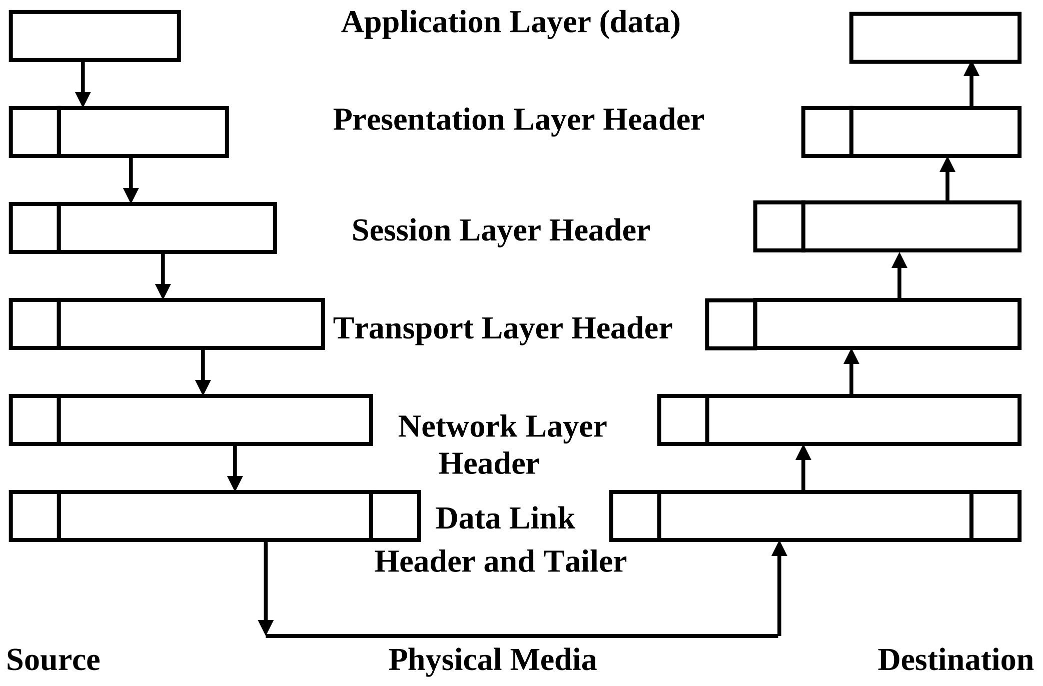


Fig. (2.3) Packaging of data in layers

It is possible at any of these layers that the message may need to be divided into different parts. At each layer a header or tailer may be attached to the message. If the message is split, each portion is treated as its own entity by all layers below the split. Each portion may then receive its own header. As the message is received at the destination host, it is passed back up through the layers. Each layer strips off its header, and if the message had been split into separate parts then the layer reassembles it.

The standard model of layered network architecture is the 7-layer International Standard Organization (ISO) Open Systems Interconnections (OSI) Reference Model. The lowest layer of this model is called the ***Physical Layer,*** this layer is concerned with the actual transmission of raw binary bits across a transmission medium. Layer 2, called ***Data Link*** layer, is designed to take the raw bits that have been transmitted and turn them into what appears to be an error-free line. The next layer is the ***Network Layer***; this layer is concerned with controlling the subnet. The key issue at this layer is routing. The ***Transport Layer*** is the fourth layer whose purpose is to provide transmission services to the higher levels without being concerned about cost-effective data transfer. It is also insures that all pieces are received correctly. The fifth layer is the ***Session Layer*** which provides away for higher level entities to establish synchronized dialogue (sessions). The sixth layer, ***Presentation Layer***, provides certain services that are frequently used by the seventh layer. The highest level, layer 7, is the ***Application Layer*** which is concerned with a number of different protocols.

In practice, the entire OSI model is not implemented. The most common layered set of protocols in use is TCP/IP suite of protocols. TCP/IP works in a very similar manner to the OSI model in that it takes a layered approach to providing network services. Each layer in the TCP/IP model communicates with the layers above and below it in the same way that the layers in the OSI model do [FIS00].

2.3 The TCP/IP Internet Layering Model [COM95]

The ISO model can be mapped to describe the TCP/IP layering scheme. TCP/IP software is organized into four conceptual layers as shown in Figure (2.4). These four conceptual layers are:

1. ***Application Layer.*** Users invoke application programs that access services available across a TCP/IP internet. Each application program may interact with one of the transport level protocols to send or receive data. Each application program chooses the style of transport needed which can be either a sequence of individual messages or a continuous stream of bytes.

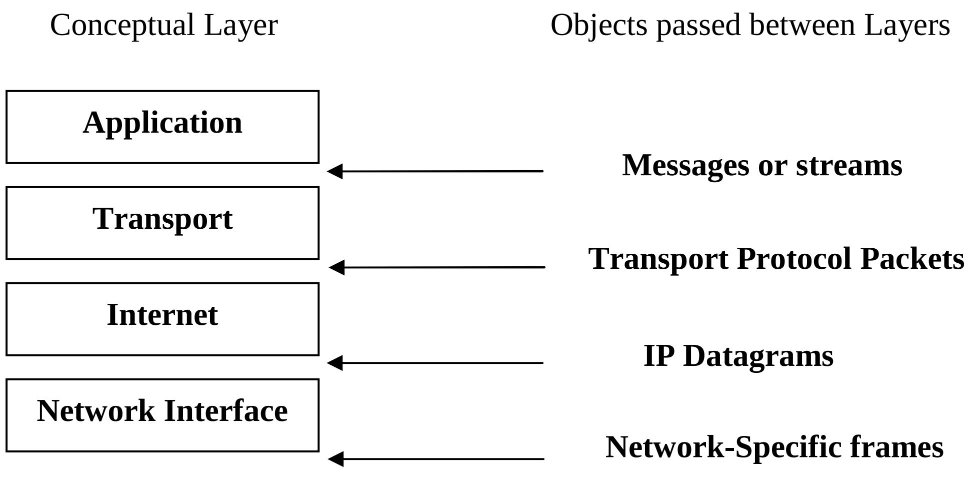


Fig. (2.4) The four conceptual layers of TCP/IP software as well as the form of data as it passes between them.

2. ***Transport Layer.*** The primary duty of transport layer is to provide communication from one application program to another. Such communication is often called end-to-end. The transport layer may regulate the flow of information. It may also provide reliable transport, ensuring that data arrives without error and in sequence.
3. ***Internet Layer.*** The Internet layer handles communications from one machine to another. It accepts a request to send a packet from the transport layer along with an identification of the machine to which

the packet should be sent. It encapsulates the packet in an IP datagram, fills in the datagram header, uses the routing algorithm to determine whether to deliver the datagram directly or send it to a router, and passes the datagram to the appropriate network interface for transmission.

4. ***Network Interface Layer.*** It is the lowest level of TCP/IP software. It comprises a network interface layer, responsible for accepting Internet Protocol (IP) datagrams and retransmitting them over a specific network.

2.4 TCP/IP Suite

The generic term "TCP/IP" usually means anything and everything related to the specific protocols of TCP and IP. It can include other protocols, applications, and even the network medium. Samples of these protocols are: User Datagram Protocol (UDP), Address Resolution Protocol (ARP), and Internet Control Message Protocol (ICMP). Samples of the related applications are: TELNET, File Transfer Protocol (FTP), and Network File System (NFS) [RFC1180].

The TCP/IP suite is not a single protocol; rather, it is four layers communication architecture that provide some reasonable network features, such as end-to-end communications, packet sequencing, internetwork routing, and some specialized functions unique to US Department of Defense communications needs (such as standardized message priorities).

The bottom layer, ***network services***, provides data for communication through the network hardware (such as Ethernet, Token Ring, and Asynchronous Transfer Mode (ATM)). The layer above the network services layer is referred to as the ***internet protocol*** (IP) layer. The

IP layer is responsible for providing a datagram service that routes data packets between dissimilar network architectures. IP has a few interesting features, one of which is the reliability. As a datagram service, IP does not guarantee delivery of data. Data concurrency, sequencing and delivery guarantee is the job of TCP protocol. TCP provides error control, retransmission, packet sequencing, and many other capabilities. The structure of TCP/IP protocol set is shown in Figure (2.5) along with the approximately equivalent ISO model layers.

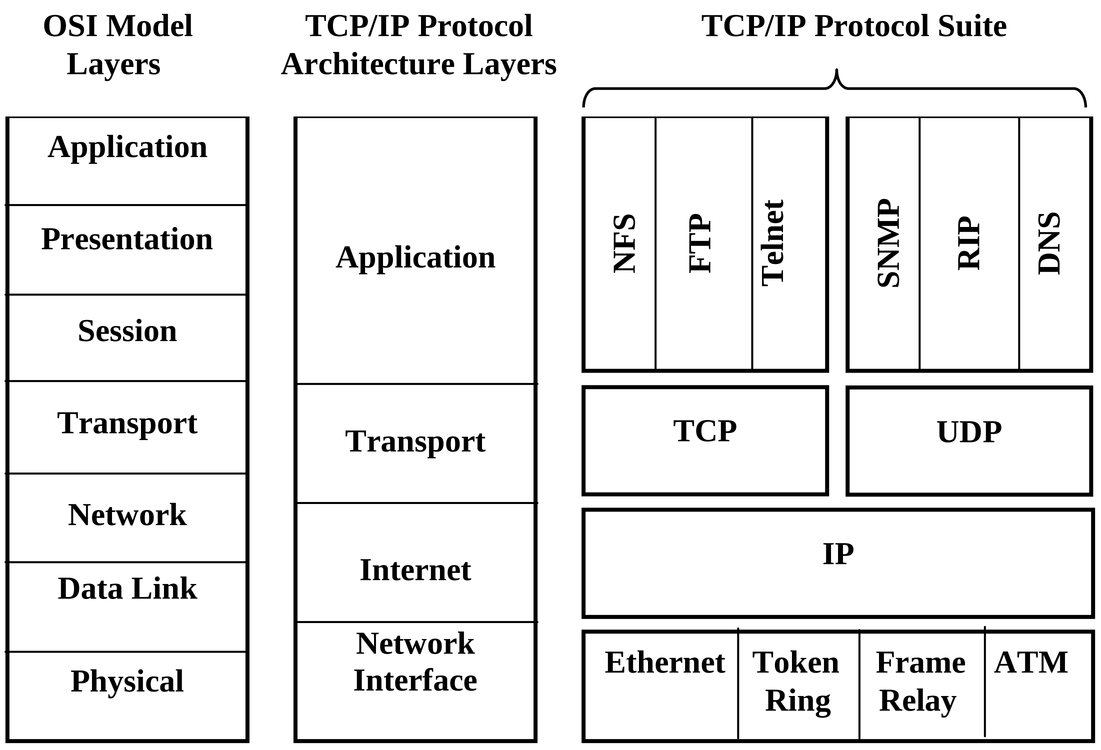


Fig. (2.5) TCP/IP protocol relationships

It can be seen that TCP/IP is essentially a four layer model, although the layers have no clear cuts as in the OSI model. The TCP/IP model has been drawn according to its use, rather than being defined first and then the protocols specified. In OSI protocols, every thing appears to be put into the protocol, but parts are made optional. In TCP/IP, the protocols are kept very simple, if more functionality is required then another protocol is added to deal with the situation [JAI97].

2.4.1 Internet Protocol

The internet protocol (IP) is the network layer protocol which is the glue that holds the whole Internet together. Unlike most older network layer protocols, it was designed from the beginning with internetworking in mind. Its job is to provide a best (i.e. not guaranteed) way to transport datagrams from source to destination, without regard to whether these machines are on the same network or whether there are other networks in between them [TAN03].

The IP provides the capability of fragmentation and reassembly of long datagrams, if necessary, for transmission through “small packet” networks. The internet protocol is specifically limited in scope to provide the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. There is no mechanism to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols.

The internet protocol implements two basic functions: addressing and fragmentation. The internet modules use the addresses carried in the IP header to transmit internet datagrams toward their destinations, while it use fields in the IP header to fragment and reassemble internet datagrams when necessary for transmission through “small packet” network. The IP uses four key factors in providing its service: Type of Service, Time to Live, Options, and Header Checksum. The detected errors may be reported via the internet control message protocol (ICMP) which is implemented in the IP module [RFC791].

An IP datagram consists of header part and text part. The header has 20-byte fixed part and a variable length optional part [TAN03]. A summary of the contents of the internet header is shown in Figure (2.6): The fields of this header are [RFC791]:

Ver: The Version (4 bits) field indicates the format of internet header.

IHL: The Internet Header Length(4 bit) in words, each word is 32 bits. The minimum value for a correct header is 5.

TOS: Type Of Service (8 bits) field provides an indication of the abstract parameters of the quality of the desired service.

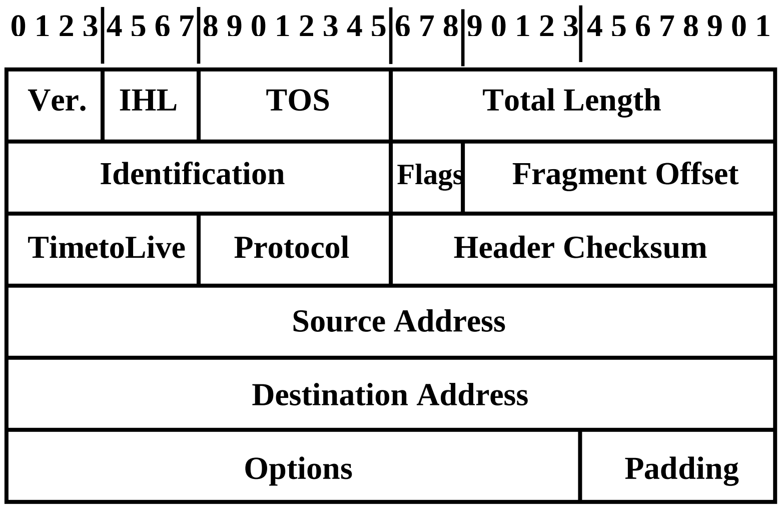


Fig. (2.6) Internet datagram header

Total Length: Its length is 16 bits which is the length of datagram, measured in octets, including internet header and data.

Identification: Its length is 16 bits, it an identifying value assigned by the sender to aid in assembling the fragment of a datagram.

Flags: Its length is 3 bits, it holds various control flags.

Fragment Offset: Its length is 13 bits, this field indicates where in the datagram this fragment belong.

Time to Live: Its length is 8 bits, this field indicates the maximum time the datagram is allowed to remain in the internet system.

Protocol: Its length is 8 bits, this field refers to the next level protocol used in the data portion of the internet datagram.

Header checksum: Its length is 16 bits field, this field holds the checksum value of the header only.

Source Address: its length is 32 bits, this field indicates the IP address of source.

Destination Address: its length is 32 bits, it refers to the IP address of destination.

Options: this field is variable in length; it may be zero or more. In some environments the security option is required in all datagrams.

Padding: this field is variable in length, it is used to ensure that the internet header ends on a 32 bit boundary, the padding is zero.

2.4.2 Transmission Control Protocol

Transmission Control Protocol (TCP) was specifically designed to provide a reliable end-to-end byte stream over an unreliable internetwork. An internetwork differs from a single network because different parts may have widely different topologies, bandwidths, delays, packet sizes, and other parameters. TCP was designed to dynamically adapt to properties of the internetwork and to be robust in the face of many kind of failures [TAN03].

It is important to understand that TCP is a communication protocol, not a piece of application software [COM00]. TCP is assumed to be a module in an operating system. The users access TCP much like they would access the file system. The TCP may be called by other operating system functions (for example, to manage data structures). The actual interface to the network is assumed to be controlled by a device driver module. The TCP does not called by the network device driver directly, but rather it is called by the internet datagram protocol module, which may in turn called by the device driver.

Transmission is made reliable via the use of sequence numbers and acknowledgments. Conceptually, each segment is assigned a sequence number. Segments also carry an acknowledgment number which is the sequence number of the next segment. An acknowledgment by TCP does not guarantee that the data has been delivered to the end user, but only that the receiving TCP has taken the responsibility to do so. To govern the flow of data between TCPs, a flow control mechanism is employed. The receiving TCP reports a "window" to the sending TCP. This window specifies the number of octets, starting with the acknowledgment number, that the receiving TCP is currently prepared to receive.

The format of TCP header is shown in Figure (2.7); it includes the following fields [RFC793]:

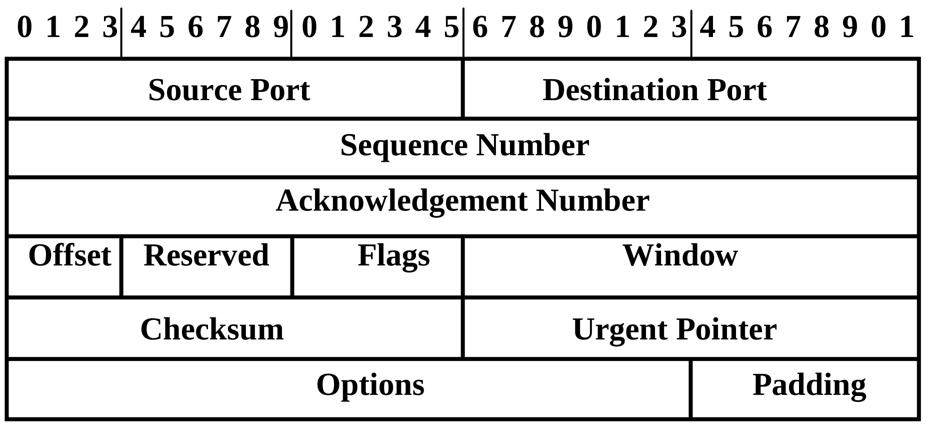


Fig. (2.7) TCP header

Source Port: Its length is 16 bits, it is the source port number.

Destination Port: Its length is 16 bits, it is the destination port number.

Sequence Number: Its length is 32 bits, it refers to the sequence number.

Acknowledgment Number: Its length is 32 bits, Sequence and Acknowledgment Numbers are used for windowing acknowledgment technique.

Data Offset: Its length is (4 bits), is refers to the number of 32 bit words in the TCP header, it indicates where the data begins.

Reserved: Its length is 6 bits, reserved for future use, must be zero.

Flags or Control Bits: (6 bits), there are several bits used as status indicators to show, for example, the resetting of the connection.

Window: Its length is 16 bits, it refers to the number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.

Checksum: Its length is 16 bits, it is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text.

Urgent Pointer: Its length is 16 bits, this field holds the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the sequence number of the octet following the urgent data. This field is only be interpreted in segments with the URG control bit set.

Options: This variable sized field contains some negotiation parameters, to set the size of the TCP packets for example. Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum.

2.5 Ports, Connections, and Endpoints [COM00]

TCP allows multiple application programs on a given machine to communicate concurrently, and it demultiplexes incoming TCP traffic among application programs. TCP uses protocol port numbers to identify the ultimate destination within a machine, each port number assigned to a small integer used to identify it. TCP ports are complex because a given port number does not correspond to a single object, instead TCP has been build on the connection abstraction, in which the object to be identified are virtual circuit connections, not individual ports.

Since TCP uses the connection, not the protocol port; connections are identified by a pair of endpoints. TCP defines an endpoint to be a pair of integers (*host, port*), where *host* is the IP address for a host and *port* is a TCP port on that host. For example, the endpoint (*128.10.2.3, 25*) specifies TCP port 25 on the machine with IP address *128.10.2.3*, and if there is a connection form machine (*128.10.2.3*) to machine (*128.10.2.7*), it might be defined by the endpoints: (*128.10.2.3, 1069*) and (*128.10.2.7, 25*).

2.5.1 TCP Port 139

Port 139 is Network Basic Input Output System (NetBIOS) Session Service, it is used for resource sharing on Windows 9x, ME and NT [BIN07].

A resource sharing computer network is defined to be a set of independent and interconnected computer systems, so as to permit each computer system to utilize all of the resources of each other computer systems. That is, a program running in one computer system should be able to call on the resources of the other computer systems much as it would normally call a subroutine [RFC61].

The goal of resource sharing is to make all programs, equipment, and especially data available to any one on the network without regard to the physical location of the resource and the user. An obvious and widespread example is having a group of office workers share a common printer [TAN03].

The systems which are trying to connect to share files that might be available on another system, hence these shared files should be blocked because most of these traffics are result of worms or viruses which can use open file shares to propagate, they also can be the result of malicious users attempt to connect to any computer. Once these systems connected, they can download, upload or even delete or edit files on the connected file share.

If the systems used open file shares (including sharing of printers, etc) on a local network (LAN), then it should be using a firewall such that the local file shares are not accessible from the internet because connecting to open file shares is likely the easiest and most common hack on the internet [BIN07].

Port 139 is the single most dangerous port on the Internet. All "File and Printer Sharing" on a Windows machine runs over this port. About 10% of all users on the Internet leave their hard disks exposed on this port. This is the first port hackers want to connect to, and the port that firewalls block [IBM07].

The NetBIOS service has become the dominant mechanism for personal computer networking. NetBIOS provides a vendor independent interface for the IBM Personal Computer (PC) and compatible systems. NetBIOS defines a software interface not a protocol. There is no "official" NetBIOS service standard. In practice, however, the IBM PC-Network version is used as a reference.

NetBIOS has generally been confined to personal computers to date. However, since larger computers are often well suited to run certain NetBIOS applications, such as file servers, this specification has been designed to allow an implementation to be built on virtually any type of system where the TCP/IP protocol suite is available. NetBIOS was designed for use by groups of PCs. Both connection and connectionless services are provided, and broadcast and multicast are supported. NetBIOS applications employ NetBIOS mechanisms to locate resources, establish connections, send and receive data with an application peer, and terminate connections, these mechanisms will collectively be called the NetBIOS Service [RFC1001].

2.5.2 TCP Port 445

TCP port 445 is a new port used by Windows 2000, Windows XP and Windows Server 2003 and used for Server Message Block (SMB) over TCP. The SMB protocol is used among other things for file sharing in Windows NT/2000/XP. In Windows NT it ran on top of NetBIOS over TCP/IP (NetBT), which uses the ports 137, 138 (UDP) and 139 (TCP). In Windows 2000/XP/2003, Microsoft added the possibility to run SMB directly over TCP/IP, without the extra layer of NetBT, to do this they use TCP port 445. NetBIOS on WAN or over the Internet, however, may lead to an enormous security risk. All sorts of information (such as the domain, workgroup and system names, as well as account information) are obtainable via NetBIOS [DAN07].

If the client has NetBT enabled, it will always try to connect to the server at both port 139 and 445 simultaneously. If there is a response from port 445, it sends a reset to port 139, and continues its SMB session to port 445 only. If there is no response from port 445, it will continue its SMB session to port 139 only, if it gets a response from there. If there is no response from either of the ports, the session will fail completely. If the client has NetBT disabled, it will always try to connect to the server at port 445 only. If the server answers on port 445, the session will be established and continue on that port. If it doesn't answer, the session will fail completely. If the server for example runs Windows NT 4.0 and has NetBT enabled, it listens on UDP ports 137, 138, and on TCP ports 139, 445. If it has NetBT disabled, it listens on TCP port 445 only [VID04].

2.6 Server Message Block (SMB) Protocol

At (1984), IBM created SMB [COD07] which is an important protocol because of the large number of PCs out there that already have client and server implementations running on them. All Windows for Workgroups, Windows 95 and Windows NT systems are capable of running SMB as a client, a server, or both. SMB, which stands for Server Message Block, is a protocol for sharing files, printers, serial ports, and communications abstractions such as named pipes and mail slots between computers. SMB is a client-server, request-response protocol. Figure (2.8) illustrates the way in which SMB works.

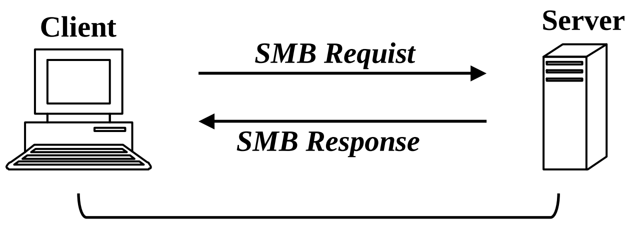


Fig. (2.8) SMB work

Servers make file systems and other resources (printers, mailslots, APIs) available to clients on the network. Client computers may have their own hard disks, but they also want access to the shared file systems and printers on the servers.

Clients connect to servers by using NetBIOS over TCP/IP, NetBIOS Enhanced User Interface (NetBEUI) or Internetwork Packet Exchange/Sequenced Packet Exchange (IPX/SPX). SMB can run over multiple protocols as shown in Figure (2.9).

OSI					**TCP/IP**
Application	**SMB**				**Application**
Presentation					
Session	**NetBIOS**	**NetBEUI**	**NetBIOS**	**NetBIOS**	
Transport	**IPX**		**DECnet**	**TCP&UDP**	**TCP/UDP**
Network				**IP**	**IP**
Link	**802.2,2 802.3,802.5**	**802.2,2 802.3,802.5**	**EthernetV2**	**EthernetV2**	**Ethernet or others**

Fig. (2.9) SMB over multiple protocols

SMB can be used over TCP/IP, NetBEUI and IPX/SPX. If TCP/IP or NetBEUI are in use, then the NetBIOS API is being used. Once they have established a connection, clients can then send commands (SMBs) to the server that allow them to access shares, open files, read and write files, and generally do all the sort of things that user want to do with a file system.

However, in the case of SMB, these things are done over the network. SMB was also sent over the Digital Equipment Corporation network protocol (DECnet). Since the inception of SMB, many protocol variants have been developed to handle the increasing complexity of the environments that it has been employed in [SHA07].

Microsoft and Intel created the first rendition of the SMB/CIFS file sharing protocol called Core Protocol which was the first protocol variant [COD07], known to SMB implementations as PC NETWORK PROGRAM

1.0. Subsequent variants were introduced when more functionality was needed. Core protocol could handle a basic set of operations, including [SHA07]:

1. Connecting to and disconnecting from file and print shares.
2. Opening and closing files.
3. Opening and closing print files.
4. Reading and writing files.
5. Creating and deleting files and directories.
6. Searching directories.
7. Getting and setting file attributes.
8. Locking and unlocking byte ranges in files.

The protocol elements (requests and responses) that clients and servers exchange called SMBs. They have a specific format that is very similar for both requests and responses. Each consists of a fixed size header portion, followed by a variable sized parameter and data portion. After connecting at the NetBIOS level, the client is ready to request services from the server. However, the client and server must first identify which protocol variant they each understand [SHA07].

The actual protocol variant client and server will use is negotiated using the *negprot* SMB which must be the first SMB sent on a connection [SHA07]. The Negotiate SMB command client request is used to resolve the protocol dialect for a session. The client sends a list of dialects. The server responds with the index number in the list of an acceptable dialect. The Negotiate SMB command packet defines the data portion of the CIFS client request and server response packets for the command code SMB_COM_NEGOTIATE [MIC06]. As shown in Figure (2.10), the client sends a Negotiate SMB (*negprot*) to the server, listing the protocol dialects that it understands.

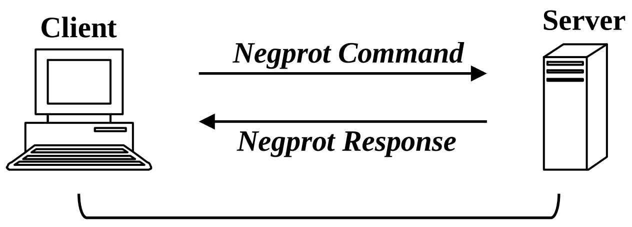


Fig. (2.10) SMB Negotiate

The server responds with the index of the dialect that it wants to use, or 0xFFFF if none of the dialects was acceptable. Once a protocol is established, the client can proceed to logon to the server if required. They do this with a Session setup and X (*sesssetupX)* SMB. The response indicates whether or not they have supplied a valid username password pair and if so, can provide additional information. One of the most important aspects of the response is the User ID (UID) of the logged on user. UID is a field in SMB header, this UID must be submitted with all subsequent SMBs on that connection to the server.

In older variant protocols (Core and Core Plus) the client cannot logon, while in CIFS protocol he can. Once the client has logged on, the client can proceed to connect to a tree. The client sends a Tree connect SMB (*tcon)* or Tree connect and X SMB command *(tconX)* specifying the network name of the share that they wish to connect to, and if all is acceptable, the server responds with a Tree ID (TID), TID is a field in SMB header. The client will use TID in all future SMBs relating to that share, as shown in Figure (2.11).

Having connected to a tree, the client can now open a file with an open SMB, followed by reading it with read SMBs, writing it with write SMBs, and closing it with close SMBs [SHA07].

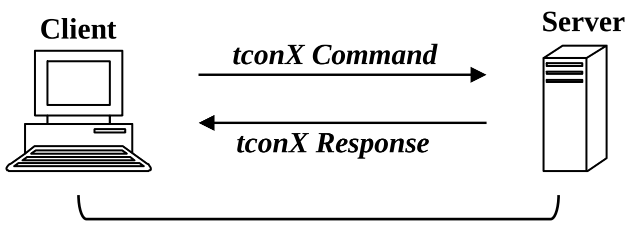


Fig (2.11) SMB Tree Connect

2.7 Common Internet File System (CIFS) Protocol

CIFS is a network protocol whose most common use is sharing files on a Local Area Network (LAN). At (1996) SMB was renamed as Common Internet File System (CIFS) with new features [COD07]. Key features that CIFS offers are [JAV07]:

1. ***File Access with integrity:*** CIFS supports the usual set of file operations; open, close, read, write and seek. CIFS also supports file and record lock and unlocking. CIFS allows multiple clients to access and update the same file while preventing conflicts by providing file sharing and file locking.
2. ***Optimization for Slow Links:*** The CIFS protocol has been tuned to run well over slow-speed dial-up lines. The effect is improved performance for users who access the Internet using a modem.
3. ***Security:*** CIFS servers support both anonymous transfers and secure, authenticated access to named files. File and directory security policies are easy to administer.
4. ***Performance and Scalability:*** CIFS servers are highly integrated with the operating system, and are tuned for maximum system performance. CIFS supports all Microsoft platforms after Windows 95. It also supports other popular operating systems such as Unix, IBM LAN server etc.

5. ***Unicode File Names:*** File names can be in any character set, not just character sets designed for English or Western European languages.
6. ***Global File Names:*** Users do not have to mount remote file systems, but can refer to them directly with globally significant names, instead of ones that have only local significance.
7. ***CIFS complements Hypertext Transfer Protocol (HTTP):*** Providing more sophisticated file sharing and file transfer than older protocols, such as FTP.

The CIFS protocol works by sending packets from the client to the server. Each packet is typically a basic request of some kind of file access (such as open, close or read file). The server then receives the packet, checks to see if the request is legal, verifies the client has the appropriate file permissions, and finally executes the request and returns a response packet to the client. The client then parses the response packet and can determine whether or not the initial request was successful.

CIFS is a fairly high-level network protocol. In the OSI model, it is probably best described at the Application/ Presentation layer. This means CIFS relies on other protocols for transport. The most common protocol used for reliable transport is NetBT. Many different protocols have been used for the transport layer, however with the enormous popularity of the Internet NetBT has become the de-facto standard. Although file sharing is CIFS's primary purpose, there are other functions that CIFS is commonly associated with. Most CIFS implementations are also capable of determining other CIFS servers on the network (for browsing), printing, and even complicated authentication techniques [COD07].

2.7.1 The Common Internet File System Protocol Uses

CIFS protocol is most commonly used with Microsoft operating systems. Windows for Workgroups was the first Microsoft operating system had used CIFS, and since then each newly issued Microsoft operating system is capable to function as both a CIFS server and client. Microsoft operating systems use CIFS for remote file operations (typically mapping network drives), browsing (via the Network Neighborhood icon), authentication (NT and Windows 2000), and remote printer services. It would be fair to say that the core of native Microsoft networking is built around its CIFS services. Because of Microsoft large corporate and home user base, CIFS protocol is found virtually everywhere. Flavors of Unix operating system also implement a CIFS client/server via Samba program. Apple computers also have CIFS clients and servers available, which had make CIFS the most common and available protocol for file sharing [COD07].

CIFS is based on the file system used on Microsoft Networks. It is being targeted by Microsoft to replace the file system services of both File Transfer Protocol (FTP) and Network File System (NFS), and allows to directly mount CIFS remote system as a directory or drive, hence it can transfer files to or from remote site just by using a regular copy command or file manager. The advantages of CIFS can be better demonstrated if it is compared with two other internet file transfer protocols (like FTP and NFS). [BIZ07]

2.7.2 Common Internet File System Packet Header

Every CIFS request and response uses the packet header as shown in Figure (2.12) [COD07]:

Header: The beginning of every CIFS packet contains a 4-byte header. The first byte is 0xFF, followed by the ASCII representation of the letters S, M, and B.

Command: The command field contains a one-byte code indicating the CIFS packet type. Examples from the CIFS1.0 draft for this field are SMB_COM_READ_ANDX, (0x2E), SMB_COM_TREE_CONNECT (0x70), and SMB_COM_NEGOTIATE (0x72).

Error class: A server flag indicates whether or not a given request was successful. Typically, when the field is zero it indicates the request success. If non-zero, the field indicates what class the error code is from. The error class field takes one of the following values:

1. ERRDOS (0x01): Error is from the core of DOS operating system.
2. ERRSRV (0x02): Error is generated by the server network file manager.
3. ERRHRD (0x03): Hardware error.
4. ERRCMD (0xFF): Command was not in the SMB format

Error code: This 16-bit field indicates the type of error that has occurred. When its value is zero it indicates no error. If its value is non-zero then this number in conjunction with the error class above can be looked up in the CIFS1.0 draft to give full error descriptions (such as, bad password or file does not exist). As with the error class, this field is set only by servers in response to a previous request.

0 1 2 3 4 5 6 7	**0 1 2 3 4 5 6 7**	**0 1 2 3 4 5 6 7**	**0 1 2 3 4 5 6 7**
0xFF	**S**	**M**	**B**
Command	**Error Class**	**Must be zero**	**Error Code**
Error Code	**Flags**	**Flags2**	

Pad or security signature typically pad and therefore must be zero		
Tree ID (TID)		**Process ID (PID)**
User ID (UID)		**Multiplex ID (MID)**
WordCount	**ParameterWords [WordCount]**	
ByteCount		
Buffer [ByteCount]		

Fig. (2.12) The structure of CIFS header

Flags: Most of the 8 bits in the flags field specify particular options. The only one of interest is bit-3. When bit-3 is set to 1, all pathnames in this particular packet must be treated as caseless (case insensitive). While, when it is set to 0, all pathnames are case sensitive.

Flags2: This 16-bit field specifies more options. Bits that are useful: Bit 0, if set, indicates that the server may return long file names in the response. Bit 6, if set, indicates that any pathname in the request is a long file name. Bit 15, if set, indicates strings in the packet are encoded as UNICODE.

Pad/security signature: This field is typically set to zero.

Tree ID (TID): It is a 16-bit integer identifies which resource (disk share or printer, typically) this particular CIFS packet is referring to. When packets are exchanged which do not have anything to do with a resource, this number is meaningless and ignored.

Process ID (PID): It is a 16-bit integer identifies which process is issuing the CIFS request on the client. The server uses this number to check for concurrency issues (typically to guarantee that files will not be corrupted by competing client processes).

User ID (UID): It is 16-bit number identifies the user who had issued CIFS request on the client side. The client must obtain the UID from the server by sending a CIFS session setup request containing a username and a password. A UID is valid only for the given NetBIOS session. Other sessions could potentially be using an identical UID that the server correlates with a different user.

Multiplex ID (MID): It is a 16-bit integer used to allow multiple outstanding client requests to exist without confusion.

WordCount and parameter words: CIFS packets use these two fields to hold command-specific data. The *wordcount* specifies how many 16-bit words the parameter words field will actually contain. In this way, each CIFS packet can be adjusted to the size needed to carry its own command-specific data. The *wordcount* for each packet type is typically constant. There are two *wordcounts* defined for every single command; one *wordcount* for the client request and another for the server response. These two counts are needed because the amount of data necessary to make a request is not necessarily the same amount needed to issue a reply.

ByteCount and buffer: These fields are very similar to the *wordcount* and parameter words field mentioned above; they hold a variable amount of data that is specified on a per packet basis. The *bytecount* indicates how many bytes of data will exist in the buffer field that follows.

2.7.2.1 Network Create andX Command

Network Create and X (NTcreate_andX) is a CIFS Command packet that is held when there is an access to resource sharing, Figure (2.13) shows the client request of NTcreate_andX header, while Table (2.1) lists the details of NTcreate_andX header fields. [MIC06]

0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	0 1 2 3 4 5 6 7
WordCount	AndXCommand	AndXReserved	AndXOffset
	Reserved	NameLength	Flags
			RootDirectoryFid
			DesiredAccess
			AllocationSize
			FileAttributes
			ShareAccess
			CreateDisposition
			CreateOptions
			ImpersonationLevel
			SecurityFlags
Byte Count		File Name	

Fig. (2.13) NTcreate_andX header

Table (2.1) the details of NTcreate_andX header fields

Name of Field	**Size (Byte)**	**Description**
WordCount	1	Refers to count of parameter words.
AndXCommand	1	It specified for secondary command
AndXReserved	1	Reserved. This value must be 0 (zero).
AndXOffset	1	This field is offset to *WordCount* location for the following command.
Reserved	1	Reserved, and its value must be 0.
NameLength	1	It holds the length of the *Filename[]* field in bytes.
Flags	4	It specified for flags.
RootDirectory	4	Refers to optional directory for relative open.
DesiredAccess	8	It specified for access desired, its value illustrates in Table (2.2).

AllocationSize	8	Refers to initial allocation size.
FileAttribute	4	It specified for flags and attributes for the file.
ShareAccess	4	It specified for types of share access.
CreateDisposition	4	Flags defining the action to take if the file already exists.
CreateOptions	4	It specified for file create options.
Impersonation	4	Security QOS information. This field specifies the client impersonation level.
SecurityFlags	1	Security tracking mode flags.
ByteCount	2	Count of data bytes.
Filename[]	Variable	It specified the name of file and its path

From table (2.1), three fields are considered in this work, because they guides to the type of access, these fields are (desired access, create options, and file name). The first two fields are shows in details as follows:

- **Desired Access Field:** This field is represented by access mask. The access mask, which is one of encoding data types, is a 32-bit value containing specific, standard, and generic rights. These rights are used in access control entries, and are the primary means of specifying the requested or granted access to an object, these rights are [MIC06]:
  1. Specific rights: Values 0-15, these rights contain the access mask specific to the object type associated with the mask.
  2. Standard rights: Values 16-23, these rights contain the object's standard access rights and can be a combination of the predefined standard rights flags. Table (2.2) shows the specific and standard rights.
  3. Generic rights: Values 24-31, these rights contain the object's generic access rights and can be a combination of the predefined generic rights flags. Table (2.3) lists generic right.

Table (2.2) Specific and standard rights of access mask [MIC06]

Flag	Description
DELETE 0x00010000	Delete access.
READ_CONTROL 0x00020000	Read access to the owner, group, and discretionary access-control list (ACL) of the security descriptor.
WRITE_DAC 0x00400000	Write access to ACL.
WRITE_OWNER 0x00080000	Write access to owner.
SYNCHRONIZER 0x00100000	Windows NT synchronize access.

Table (2.3) Generic rights of access mask [MIC06]

Flag	Description
ACCESS_SYSTEM_SECURITY 24	Access system security. This flag is not a typical access type. It is used to indicate access to a system ACL. This type of access requires the calling process to have a specific privilege.
MAXIMUM_ALLOWED 25	Maximum allowed.
26-27	Reserved.
GENERIC_ALL 28	Generic all.
GENERIC_WRITE 30	Generic write.
GENERIC_READ 31	Generic read.

- **Create Options Field:** It specified for options to create the files. The value of this field illustrates in Table (2.4).

Table (2.4) Flags for files

Value	Meaning
FILE_SUPERSEDE 0x00000000	If the file already exists, supersede it by the specified file. Otherwise, create the file.
FILE_OPEN 0x00000001	If the file already exists, return success; otherwise, fail the operation.
FILE_CREATE	Create file.

0x00000002	
FILE_OPEN_IF 0x00000003	Open the file if it already exists; otherwise, create the file.
FILE_OVERWRITE 0x00000004	Overwrite the file if it already exists; otherwise, fail the operation.
FILE_OVERWRITE_IF 0x00000005	Overwrite the file if it already exists; otherwise, create the file.

2.7.2.2 Delete Directory Command

Delete Directory (SMB_COM_DELETE_DIRECTORY) command specified for deletes the directories from shared resources. Figure (2.14) shows the header of this command, while Table (2.5) illustrates the details of each field.

0 1 2 3 4 5 6 7	**0 1 2 3 4 5 6 7**	**0 1 2 3 4 5 6 7**
WordCount	ByteCount	BufferFormat
DirectoryName[ ]		

Fig. (2.14) Delete directory header

Table (2.5) The details of Delete directory header

Name of Field	**Size (Byte)**	**Description**
WordCount	1	Refers to count of parameter bytes. The value is 0.
ByteCount	1	Count of data bytes. The value is greater than 0.
BufferFormat	1	Buffer format. The value is 0x04.
DirectoryName[ ]	Variable	It specified directory name and its path.

2.8 Computer Security Concepts

Intrusion detection was constantly evolving to meet a set of functional goals, all associated with improving the security of computers and networks. In this section some of the fundamental terms and concepts of computer and network security are given [BAC00].

A practical definition of a *secure* computer system is “a system that can be depended upon to behave as it is expected to” [GAR96]. From this intuitive definition of security, one can infer the fundamental concepts associated with security.

A more precise formal technical definition of computer security is given in terms of the “security triad: confidentiality, integrity, and availability”. These security triad are [BAC00]:

1. ***Confidentiality*** is the requirement that “access to information be restricted to only those users authorized for that access”. Much of the work done by the government in computer security focuses on confidentiality.

2. ***Integrity*** is the requirement that “information be protected from alteration”. Integrity is especially critical in systems handling data such as medical records (imagine the impact of someone altering doctors’ orders on a patient record) or financial accounts. Many publicized Web site attacks involve breaches of integrity, in which address tables or site content are modified.

3. ***Availability*** is the requirement that “the information and system resources continue to work, and the authorized users be able to access resources when they need them, where they need them, and in the form in which they need them”. Many network-based attacks, such as “teardrop” and “ping of death,” crash servers by sending them network traffic fashioned to exploit vulnerabilities in the operating system software running on those servers. These intrusions,

which violate availability requirements, are labeled *denial of service* attacks.

A secure computer system supports all above mentioned three goals of the security triad. In other words, a secure system protects its information and computing resources from unauthorized access, tampering, and denial of service [BAC00].

2.8.1 Threat

Most security programs are driven by a desire to address a threat. A *threat* is defined as any situation or event that has the potential to harm a system. This harm can be in the form of disclosure, destruction, or modification of data; or denial of access to data, or to the system-processing resources. Major categories of threat include hackers, viruses, fire, flood, lightning strikes; the list goes on and on. From this list, it could be noticed that threats can be either internal or external to the system and can be intentional or incidental. To achieve security goals, the physical threats must be considered as well as computer threats. This may require background investigations of personnel serving in critical roles (for example, system administrators) when they have significant control over computing and information resources [BAC00].

How are threats structured in the computer security world? There are several ways to classify threat, and some involve with the source of the threat. An early model had specified the following three categories of threats [AND80]:

1. External penetrators: Unauthorized users of the system.

2. Internal penetrators: Authorized users of the system who overstep their legitimate access rights. These internal threats are divided into the following:
   A. Masqueraders: Those who appropriate the identification and authorization credentials of others.
   B. Clandestine users: Those who successfully evade audit and monitoring measures.

3. Misfeasors: Authorized users who exceed their privileges.
4. People who attempt to gain access to a system or data.
5. Programmatic threats: including software attacks such as viruses, trojan horses, and malicious Java or ActiveX applets.
6. People who probe or scan systems in search of vulnerabilities they can exploit their search results in a later attack.

2.8.2 Vulnerability

Security problems in computer systems result from vulnerabilities. *Vulnerabilities* are weaknesses in systems that can be exploited in ways that violate security policy. Vulnerabilities occur in a two ways [BAC00]:

1. ***Technical Vulnerabilities*****:** weaknesses occur in the design and implementation of the system software and hardware.
2. ***Procedural*** **or** ***Management Vulnerabilities:*** weaknesses occur in security policy, procedures, controls, configuration, or other system management areas.

Most systems have vulnerabilities of some sort, but this does not mean that the systems are too flawed to use. Not every threat results in an attack, and not every attack succeeds. Success depends on the degree of vulnerability, the strength of attacks, and the effectiveness of any countermeasures in use. If the attacks needed to exploit vulnerability are very difficult to carry out, then the vulnerability may be tolerable. If the perceived benefit to an attacker is small, then even an easily exploited vulnerability may be tolerable. However, if the attacks are well understood and easily made, and if the vulnerable system is employed by a wide range of users, then it is likely that there will be enough benefit for someone to make an attack [RFC2828].

Several rules of thumb govern the likelihood of vulnerabilities occurring in systems. The larger system is the greater likelihood of vulnerabilities. The more complex system is the greater likelihood of vulnerabilities. The more dynamic system and its environment (for example, a system that is repeatedly updated or replaced with a new operating system), is the greater likelihood of vulnerabilities. Although threat and vulnerabilities are intrinsically related, they are not the same. Threat is the result of exploiting one or more vulnerabilities. Any intrusion detection should be designed to identify and respond to both [BAC00].

2.8.3 Intrusion Detection

During the last 27 years ago intrusion Detection (ID) had been an active field of research, its importance had attracted the scientific attention since (1980) due to the John Anderson’s publication “*Computer Security Threat Monitoring and Surveillance*”, which was one of the earliest papers in the field [MCH00].

ID is the process of monitoring computer networks and systems for violations of security policy. In simplest terms, intrusion detection systems consist of three functional components [BAC00]:

1. An information source that provides a stream of event records.
2. An analysis engine that finds signs of intrusions.
3. A response component that generates reactions based on the outcome of the analysis engine.

The first component for ID is the data source. This element can also be considered an event generator [BAC00].

The IDSs can be classified according to data sources into two types [MCH00]:

1. ***Network-Based Systems:*** Look at packets on a network segment. While network-based systems can simultaneously monitor numerous hosts, they can suffer from performance problems, especially with increasing network speeds. Many network-based systems are popular because they are easy to deploy and manage and have little or no impact on the protected system's performance.

2. ***Host-Based Systems*****:** Operate on the protected host, inspecting audit or log data to detect intrusive activity. Host-based systems can monitor specific applications in ways that would be difficult or impossible in a network-based system.

The second component in the intrusion detection process is the analysis engine, this system component takes information from the data source and examines the data for symptoms of attack or other policy violations. In intrusion detection, most analysis approaches involve misuse detection, anomaly detection or some mix of the two. The two main analysis approaches are [BAS00]:

1. ***Misuse Detection:*** Engines look for something defined to be "bad." To do this, they filter event streams, searching for activity patterns

that match a known attack or other violation of security policy. Misuse detection uses pattern-matching techniques, matching against patterns of activity known to indicate problems. Most current, commercial intrusion detection systems utilize misuse detection techniques.

2. ***Anomaly Detection:*** Engines look for something rare or unusual. They analyze system event streams, using statistical techniques to find patterns of activity that appear to be abnormal. This approach reflects the view of some intrusion detection researchers that intrusions are some subset of anomalous activity.

The third component in the intrusion detection process is response, intrusion detection system responses can be classified as [BAC00]:

1. ***Active Responses:*** The system automatically (or in concert with the user) blocks or otherwise affects the progress of the attack. Active responses involve taking action based on the detection of an intrusion.
2. ***Passive Responses:*** The system simply reports and records the problem. Passive responses are those provide information about the user, relying on the user to take subsequent action. Passive responses are important, and in many cases represent the sole response form for the system.

2.8.4 Network Monitoring

At (1990) [SHE07] a network monitoring system called "Network System Monitor (NSM)" which is developed in University of California was established to run on a SUN UNIX workstations. Before this time, most intrusion detection systems utilize the information extracted from operating system audit trails or keystroke monitors [HER90]. NSM does not use audit trails to perform its intrusion detection. Instead, it monitors the broadcast channel to observe all network traffic. As a result of this, it can monitor a variety of hosts and operating system platforms [MUK94].

Till now, the general architecture of NSM is still reflected in many commercial intrusion detection products. NSM functioned to do the following tasks [HER90]:

1. Placing the system's Ethernet network interface card into promiscuous mode; in which each network frame generates an interrupt, thereby allowing the monitoring system to listen to all traffic, not just those packets addressed to the system.
2. Capturing network packets.
3. Parsing the protocol to allow extraction of pertinent features.
4. Using a matrix-based approach to archive and analyze the features, both for statistical variances from normal behavior and for violations of pre-established rules.

NSM was a significant milestone in intrusion detection research because it was the first attempt to extend intrusion detection to heterogeneous network environments. Also, it was one of the first intrusion detection systems to run on an operational system (i.e, the local area network of the computer science department at University of California) [HER90].

The objective of NSM is to determine if the packet flow matches a known signature. There are three kinds of signatures that are particularly important [SHE07]:

1. String signatures, the packet checking process is based on looking for a text string that indicates a possible attack.
2. Port signatures, such kind of checking is simply based on detecting and analyzing the connection attempts to some well known, frequently attacked ports.
3. Header signatures, the checking is based on detecting and analyzing the dangerous or illogical combinations in packet headers.

A network-based ID system views packet traffic on its network segment as a data source. This is usually accomplished by placing the network interface card in promiscuous mode. Although this approach is simple and powerful, but it does not always work on modern network systems. For instance, in the case of switched networks the network switch acts to isolate network connections between hosts so that a host can see only the traffic that is addressed to it. Also, data that travels via other communications media (such as, dial-up phone lines) cannot be monitored using this approach [BAC00].

2.9 File System

For most users, the file system is the most visible aspect of an operating system. It provides the mechanism for on-line storage and access to both data and programs of the operating system and to all the users of the computer system.

The file system consists of two distinct parts: a collection of files, each storing related data, and directory structure (which organizes and provides information about all the files in the system). Some file systems have a third part, partitions, which are used to separate physically or logically the large collections of directories [GAL98].

2.9.1 File Types

The most common technique for managing the file types is to include the type as part of the file name. The file name consists of two parts: a *name* and an *extension,* usually separated by a period character. In this way, the user and operating system can know from the name alone what is the type of a file. For example, in MS-DOS, a name can consist of up to eight characters followed by a period and terminated by an up-to-three character extension. The file system uses the extension to indicate the type of the file and to define the type of operations that can be done on that file. For instance, only a file with a " .com ", " .exe ", or " .bat " extension can be executed. The " .com " and " .exe " files are two forms of binary executable files, whereas a ".bat" file is a batch file containing in ASCII format, commands to the operating system [GAL98].

2.9.2 Types of File Access

The need for protecting files is due to the ability to access files. On systems that do not permit access to the files of other users, protection is not needed. Thus, one extreme would be to provide complete protection by prohibiting access, while the other extreme is to provide free access with no protection. Both of these approaches are too extreme for general use, therefore *controlled access* is needed. Protection mechanisms provide controlled access. Access is permitted or denied depending on several factors, one of which is the type of requested access. Several different types of operations may be controlled; like

1. ***Read:*** Read from the file.
2. ***Write:*** Write or rewrite the file.
3. ***Execute:*** Load the file into the memory and execute it.

4. ***Append:*** Write new information at the end of the file.
5. ***Delete:*** Delete the file and free its space for possible reuse.
6. ***List:*** List the name and attributes of the file.

Other operations, such as renaming, copying, or editing the file, may also need to be controlled. For many systems, however, these higher level functions (such as copying) may be implemented by a system program that makes lower level system calls. For instance, copying a file may be implemented simply by a sequence of read requests. In this case, a user with read access can also cause the file to be copied, printed, and so on. So, protection is provided at only the lower level.

Many different protection mechanisms have been proposed. Each scheme has its own advantages and disadvantages, and each type must be selected as appropriate according to its applicability and feasibility to the intended application. For example, a small computer system that is used by only a few members of a research group may not need the same types of protection as will a large corporate computer that is used for research, finance and personnel operations [GAL98].

2.9.3 File Protection

When information is kept in a computer system, a major concern is its protection from both physical damage (reliability) and improper access (protection). Reliability is generally provided by duplicate copies of files. Many computers have system programs that automatically (or through computer operator intervention) copy disk files to a backup storage media at regular intervals (once per day or week or month) to maintain a copy for the files.

File systems can be damaged by hardware problems (such as error in reading or writing), power surges or failures, head crashes, dirt, and temperature extremes. Files may be deleted accidentally. Bugs in the file-system software can also cause file content to be lost. Protection can be provided in many ways. For small single-user system, it might provide protection by physically removing the CD drives and locking them in the desk drawer or file cabinet. In a multi-user system, however, other mechanisms are needed [GAL98]. Such mechanisms are access controls.

2.9.4 Access Controls

Early computers had no internal controls to limit their access, and any user could interact with any portion of the desired system. In today's larger and much more complex environments, however, just because individuals have been established as authorized users does not mean that they should have access to all information contained in the system. This is the job of Access Control security services, they determine which information an individual can access. There are several methods of access controls, the most three popular are [FIS00]:

1. ***Protection Table:*** It is one of the simplest methods of access controls. The established table (i.e., protection table) contains every user (or subject) who may have a privilege to access the system, and every file (or object) stored on the system. The typical access modes which may be found in the table include Read, Write, Execute, Delete, List, as shown in Figure (2.15). A location in the table with no entry means that the user has not been granted any type of access for this file. While a protection table may seem like a simple solution to the

access control problem, in the reality it is rarely implemented in this fashion.

	File 1	**File 2**	**File 3**	**File 4**
User 1			**Read, Write**	
User 2	**Read, Write**	**Read**	**List**	
User 3		**Read, Write**		**Execute**

Fig. (2.15) A sample of protection table

The reason for this is that the created table will become extremely large and sparsely populated. Most users will have access to only their own files and not to the files created by other users. This fact may lead to a sparse table in which its most entries will be blank. Reserving space for a large, mostly empty table is a tremendous waste of space.

2. ***Capabilities Based:*** Another scheme of access controls is called a capabilities-based, this scheme could be thought of as dividing a protection table, as depicted in Figure (2.16), into rows. Associated with each user is a list of the objects the user may access along with the specific access permission for each object. This list is called a Capability list. An example of the capability list for User2 is shown in Figure (2.16).

Object	**Permissions**
File1	**R,W,--,--,L**
File 2	**R,--,--,--,L**
File 3	**--,W,--,--,--**

Fig. (2.16) Capability list for User 2

3. ***Access Control Lists:*** While capability techniques can be thought of as dividing the protection table into rows, the Access Control Lists (ACL) divide it by columns. Instead of maintaining a separate list for each subject detailing the objects that a subject has access to, ACLs are created for each object and list the subjects that have permissions to access them, as shown in Figure (2.17). There are several obvious advantages in using Access Control Lists over the other techniques; the first is the ease in which a list of all subjects granted access to specific object can be determined. The second is the ease with which access can be revoked, the owner of object can simply remove or change any entry in the ACL to revoke or change the type of access granted to any subject.

File 1	**File 2**	**File 3**
User 2 RWL	**User 2 RL**	**User 1 RWL**
	User 3 RWL	**User 2 L**

Fig. (2.17) Access Control Lists

The third advantage is saving the storage space that is used in this method. The tables do not have to list each subject for every object, just those have access permission and that mode of access is granted. The drawback of this method is the difficulty in listing in all objects lists a specific subject which has access to all of them. In this case, all of ACLs would need to be scanned to determine which contained a reference to the subject in question.

CHAPTER THREE
NETWORK SECURITY MONITORING SYSTEM DESIGN

3.1 Introduction

This chapter is dedicated to introduce the design and implementation steps of the proposed network security monitoring system. This system was designed to run under Windows operating system platforms, and it was applied on client-server LANs. The antivirus and firewalls programs may collide with the work of the proposed system.

3.2 The System Model

The general structure of the proposed network security monitoring system is illustrated in Figure (3.1).

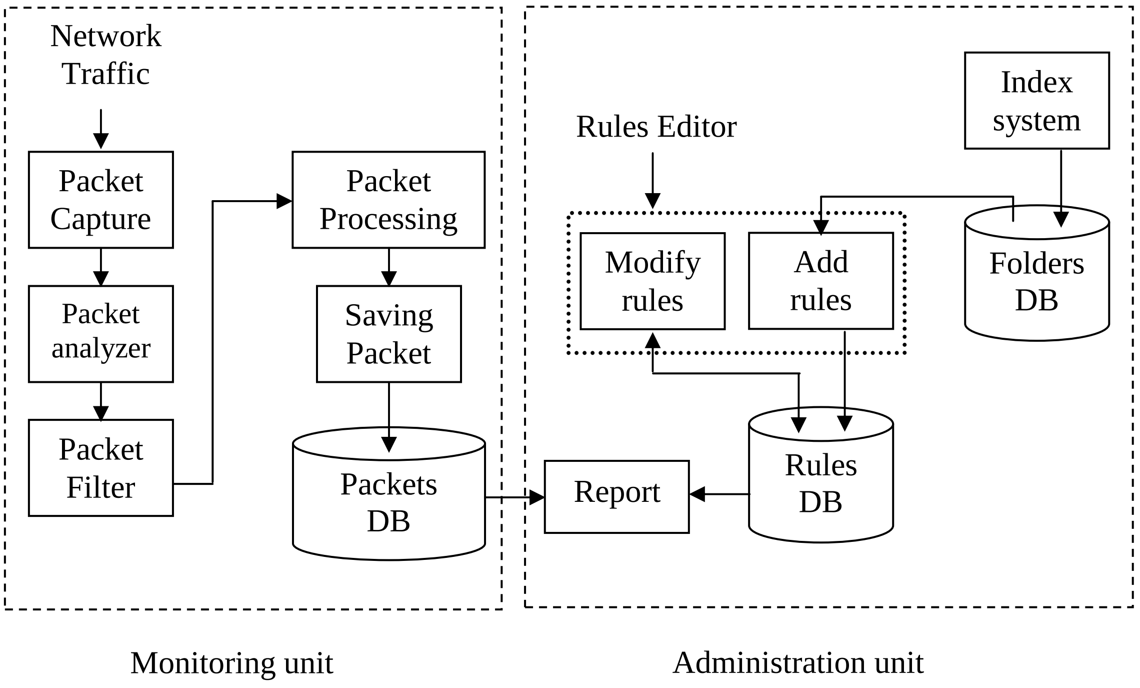


Fig. (3.1) The general system model

It is consist of two basic units: packets monitoring unit and administration unit. The monitoring unit works in client side, while the administration unit works in server machine. Monitoring unit consists of four main modules, packet capture, packet analyzer, packet processing, saving packet modules. Also, the monitoring unit has a database dedicated to archive the packets that used by administration unit. Administration unit consists of two main modules: indexing module, and rules editor. Also, the administration unit uses two main databases, one for listing the shared folders and files in the network, and the other for listing the applied rules. Rules editor consists of two main modules; *add rules* module for initializing the new rules, and *modify rules* module for modifying the existing rules; these two modules save their output (i.e., the rules) in the *rules database*.

The system begins to respond just when there is a client request. Client request means that there is row of data or packets transmitted from client to server. The monitoring module captures these packets, analyze, process, and save it in the *packets database*. On the other side (i.e., on server machine) the administration unit initializes the process of indexing the available shared folders and files by activating the *index folders* module, and saving the established index list in the *folders database*.

The output of the proposed model is a report which takes the data from two databases, packets database and rules database.

3.3 Packet Capture Module

As mentioned in chapter two, section 2.4, the TCP/IP suite is not a single protocol; rather, it has four layers communication architecture that provides some reasonable network features (such as end-to-end communications, packet sequencing, internetwork routing). These four layers of TCP/IP suite are; network layer, internet layer, transport layer and application layer.

Packet capture module captures all packets in internet layer; it means that this module captures IP packets or IP datagrams. The reason of capturing packets in this layer (not in network layer) is that “the network layer involves with hardware component”.

After starting connection, the packet capture module begins to capture packets, packet after packet, each packet figured as row of data that saved in a specific location in the memory. The next step is moving this data from its specific location in memory and save it as an array of bytes called *ReadBuffer*. Since, the maximum length of each packet is 1500 bytes therefore the length of *ReadBuffer* was assigned 1500 byte.

To easily recognize the details of each packet, the array of bytes, *ReadBuffer*, was structured as a set of fields. Packet capture module matches *ReadBuffer* structure with the internet protocol (IP) and the transmission control protocol (TCP) frames structure; they will be described in following subsections:

1. Matching the Internet Protocol Frame

The internet protocol (IP) frame structure consists of two parts: header and body (text) parts, see section 2.4.1. The header consists of 20-byte fixed part and a variable length optional part. The total length of the IP_header record is 20 bytes, its structure is list in Table (3.1).

The length of IP packet is between 40 to 1500 bytes, the first 20 bytes represent the IP header (starting from byte-0 till byte-19). Figure (3.2) illustrates the structure of IP packet and the position of IP header in the packet. After creating an IP header record, the data of *ReadBuffer* array are copied into IP header record.

Table (3.1) IP header record

Field Name	Size in bytes
Ip_verlen	1
Ip_tos	1
Ip_totallength	2
Ip_id	2
Ip_offset	2
Ip_ttl	1
Ip_protocol	1
Ip_checksum	2
Ip_srcaddr	4
Ip_destaddr	4

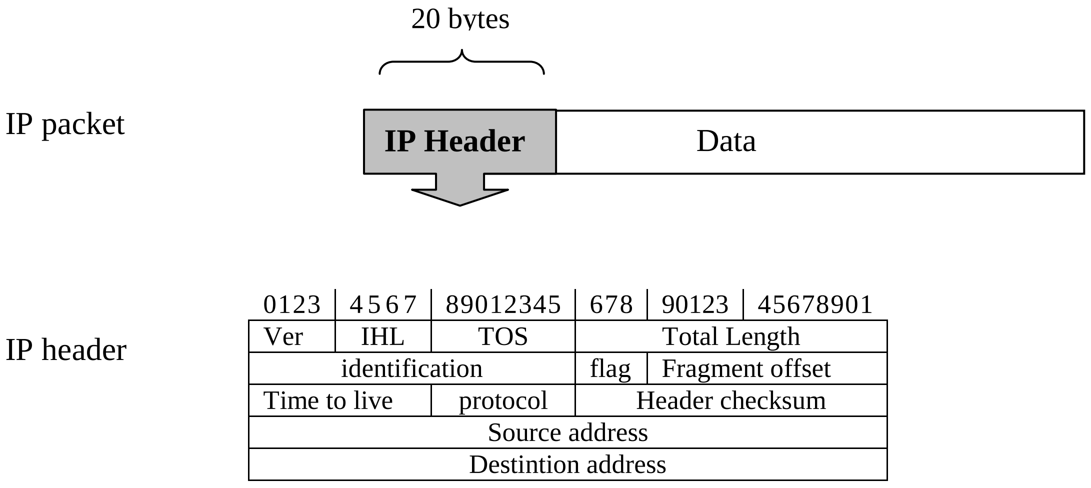


Fig. (3.2) IP packet and IP header [RFC791]

In IP header, there are some fields have length less than one byte, like *Ver* and *IHL*, to fix their values in IP record, they merged in one byte field. As shown in Table (3.1) the *ip_verlen* is defined as byte while its real length is 4-bits. In the same table, one can notice that *IHL* field was not defined in IP header record as individual field, because the fields *Ver* (4 bits) and *IHL* (4 bits) have been combined in one field called *ip_verlen* whose length is one byte.

2. Matching the Transmission Control Protocol (TCP) Frame

The TCP header consists of a 20-byte fixed part and a variable length optional part. Table (3.2) shows the structure of TCP header.

To check if the received packet is TCP or not, the *ip_protocol* field must be checked. If the field *ip_protocol* equal to 6, then the data from *ReadBuffer* (20) to *ReadBuffer* (39) are copied into TCP header record. Figure (3.3) shows the position of TCP header in IP packet.

Table (3.2) TCP header record

Field Name	***Size in bytes***
Src_portno	2
Dst_portno	2
Sequenceno	4
Acknowledgeno	4
DataOffset	1
Flag	1
Windows	2
Checksum	2
UrgentPointer	2

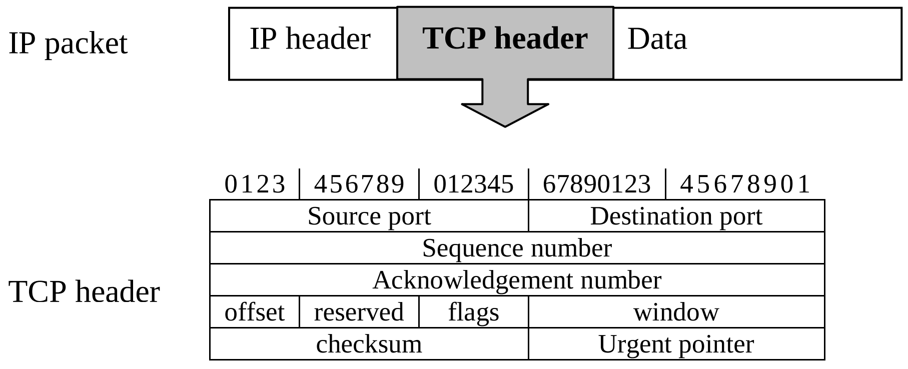


Fig. (3.3) IP packet and TCP header [RFC793]

Code list (3.1) illustrates the implemented steps to perform packet capturing task. The input is a row of data, and the output consists of IP header record, TCP header record and *ReadBuffer* array.

Code List (3.1) Packet Capture Module.

```
Input
        Row of data.
Output
        IP header record shows in Table (3.1).
        TCP header record shows in Table (3.2).
        ReadBuffer (0 to 1499) of byte.
Begin
Step 1: Start connection.
Step2: Load a row of data from receive buffer of the network card, and save it in
        specific location in the memory.
Step3: Dematerialize the data saved in specific location in the memory as an array
        (ReadBuffer) of bytes.
Step4: If length of ReadBuffer <= 0 then go to step 2, Else copy the data from
        ReadBuffer (0) to ReadBuffer (19) in IP header record.
Step5: Check if ip_protocol<>6 then go to step 2, Else copy the data from
        ReadBuffer (20) to ReadBuffer (39) in TCP header record.
Step6: If connection is not terminated then go to step2.
Step 7: End.
```

Some received packets are not necessary in the proposed system (like UDP and ICMP packets), because they doesn’t used to transfer commands to filing system. The established packet capture module ignores such kinds of packets, and restricts its analysis only on the received TCP packets.

3.4 Packets Analyzer Module

After capturing IP packets and copying some of its information in IP header and TCP header constructs, the contents of some fields should be tested. The test steps can be summarized as follows:

1. **Reading IP Header Fields:** During reading the fields of IP header, some remarks are noticed, theses remarks are:
    a. After reading and testing the fields of IP header (including version, IP header length (IHL), type of service (TOS), total length, identification, flags, fragment offset, time to live (TTL), protocol type, checksum, source IP, destination IP), the value of some fields are fixed in all received packets, as shown in Table (3.3). While, the values of other fields vary from packet to another (like total length, checksum, source IP and destination IP).

Table (3.3) The fixed fields' values of IP header

Field	Version	IHL	TOS	TTL	Protocol
Size in bits	4	4	8	16	8
Value	4 (IPv4)	5 words	Routine	128 sec.	6 (TCP) 1 (ICMP) 17 (UDP)

    b. Among all fields of IP header, two fields are very important and considered in this work, these fields are source IP and destination IP.
    c. The two merged fields (version and IHL) in one byte field called *ip_verlen*, are extracted by applying the following:

version= (ip_verlen and 240)>>4.

IHL= ip_verlen and 15.

The value of version field is extracted by capturing the highest 4 bits (i.e., highest significant nipple) from the byte *ip_verlen*, while the value of *IHL* field is extracted by getting the 4 least significant bits in ip-verlen.

2. **Reading TCP Header Fields:** During reading TCP header fields, it is found that the first two fields (source port and destination port) are important. After starting connection and trying to read a shared file from another machine, it was found that either the source or destination port will be equal to 139. Also, in some tests to access some shared files, it is found that port number 445 could be used instead of 139. Also during the analysis of TCP header, it is found that if the source port was equal to 139, then the destination port would be equal to a number greater than 1000, and it varies from run to another. Also if destination port was equal to 139, then the source port would equal to a number greater than 1000. Same behavior was noticed when port 445 is used (instead of 139). Since, the involved transport protocol in any instance of accessing shared files is TCP, and the used ports in this protocol are 139 and 445.

3. **Simulating Different Types of Filing Access:** In order to get knowledge about the role of remaining contents of the TCP packet in the access instance of shared files, some tests have been conducted by simulating various types of filing access. During these test, some read/write trials on shared files from another machine were performed. To simply searching for application protocol, all the packets that generated during these trials are registered in a capture

file of type text. Having such a text files contains all data of packet make the way of allocating the parts of packets that assign the file access type and the file name more easier.

During the investigating of the captured file contents, the following consideration and remarks are taken:

- The term “SMB” always exists in each packet, it is a part of a 4-bytes header, and the first byte is 0xFF, followed by the ASCII representation of the letters S, M, and B.
- Among the contents of the capture file, the name of the accessed files and their paths had been found. The investigation of the remaining contents of the captured packets had indicated that the used application protocol is CIFS which is a new version of SMB. The steps mentioned in code list (3.2) have been implemented to analyze only the packets that related to events of accessing shared files.

Code List (3.2): Simulating Different Types of Filing Access.

```
Input
        TCP header record
        ReadBuffer (0 to 1499) of byte.
Output
        Data_ file.txt
Begin
Step1: Apply packet capture module.
Step2: Check if Src_portno =139 or Dst_portno=139 then go to step 4.
Step3: Check if Src_portno =445 or Dst_portno=445 then go to step 4, Else go to
        step 5.
Step 4: Copy data from ReadBuffer(40) to the end of packet in Data_file.txt.
Step 5: End.
```

3.4.1 Matching the CIFS Frame Function

After capturing the CIFS packets, the CIFS header record is extracted, the elements of the record are listed in Table (3.4).

Table (3.4) CIFS Header field

Field	*Size in bytes*	*Value*
Delimiter	1	256
ID1	1	"S"
ID2	1	"M"
ID3	1	"B"
Command	1	162 or 1
Error class	1	-
Must zero	1	0
Error code	2	-
Flag	1	-
Flag2	2	-
Pad	12	0
TID	2	-
PID	2	-
UID	2	-
MID	2	-

This extraction is done by copying data from *ReadBuffer* into CIFS header record. This data laid in the area extended from element 40 to 71 in *ReadBuffer* as shown in Figure (3.4).

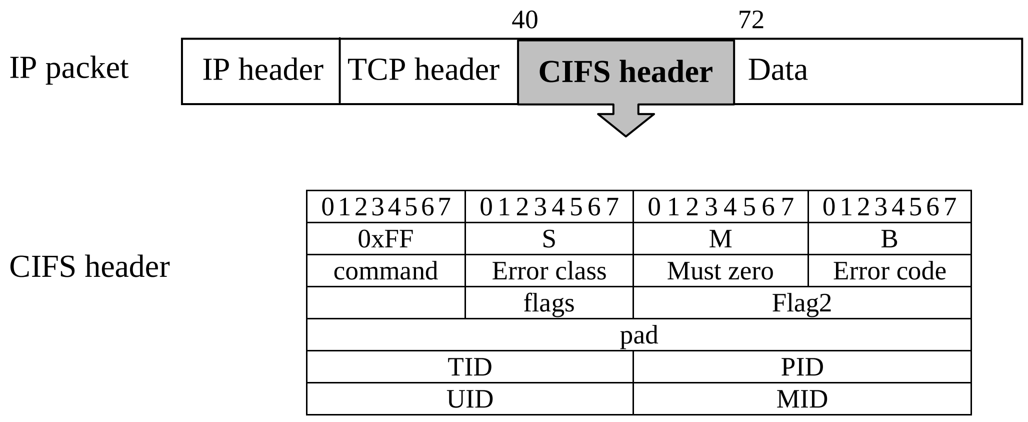


Fig. (3.4) The position of CIFS header from IP packet [COD07]

The implemented steps for extracting CIFS header record is similar to that mentioned in code list (3.2), but it different in step 4. Instead of copying data in a text file, it copy in CIFS header record.

3.5 Packet Filter Module

It was noticed that various types of packets are received at each moment, this module filters these packets and send only those filters related to file access operations for analysis. Among various types of received packets the following packets which have been filtered out are:

1. Handshake packets: such packets are used to establish a connection.
2. Broadcast packets: such kind of packets hold short messages in certain contexts, they sent by any machine and received by all the other, this type of packets uses UDP protocol.
3. ICMP packets: such packets are used to send a ping from one machine to another.
4. SMB negotiate: such packets sent by a client machine to the server, they send as the first SMB packets during the connection.
5. Session_setup_andX packets: they are SMB command packets by which the client can continue to logon to the server if required.
6. Tree_connect_andX packets: through such SMB command packets, the user can connect with the network.
7. Other SMB commands packets: include some packets of the SMB commands that are not necessary in our proposed system, like SMB_Close.
8. The repeated packets: some packets are repeatedly sent for different purposes, the most important one is for synchronization.

9. Server response packets: each command has two types of packets, client request and server response. The proposed system ignores such kind of packets.

3.6 Packets Processing Module

During capturing, pre analyzing and filtering stages, the IP header, TCP header, CIFS header and data are extracted. Packet processing module is responsible for further analyzing the packets. It takes CIFS header record with its remaining data to extract more information about the nature of the events of accessing shared files. The extracted information include: names of files, type of access, source address…etc.

The remaining contents of packet is called command packet. Command packet is identified by the *command* field in CIFS header (*CIFS.cmd*). The CIFS command packets indicate which file accessing command is hold in the captured packet (like SMB_COM_DELETE_ DIRECTORY 0x01, SMB_COM_OPEN 0x02, SMB_COM_CREATE 0x03…etc).

Generally, there are two kinds of CIFS *command* packets, either client request or server response packets. Packet processing module is concerned with the client request packet and it ignores server response packets.

The types of access that considered in this work are: Read, Write, Copy, and Delete accesses. According to these four accesses, the following four cases have been applied:

1. ***Read Access:*** The type of command packet that used in this access is *NTcreate_andX*. All CIFS commands have specific numbers, and the command number of *NTcreate_andX* equal 0xA2. Figure (2.13) shows the header of *NTcreate_andX* command. To copy the remaining data

from *ReadBuffer* to *NTcreate_andX* header, its dedicated record is created. Table (3.5) shows the fields of this record.

Table (3.5) NTcreate_andX header fields

Field	***Size in bytes***
Wordcount	1
andXcommand	1
AndXreserved	1
andXoffset	2
Reserved	1
Namelength	1
Flags	4
RootDirectoryFid	4
DesiredAccess	8
AllocationSize	8
FileAttributes	4
ShareAccess	4
CreateDisposition	4
CreateOptions	4
ImpersonationLevel	4
SecurityFlags	1
Bytecount	2
Filename	specified in Namelength field

To check if the received TCP packet is *NTcreate_andX* or not, the *CIFS_cmd* field must be checked. If the field *CIFS.cmd* equal to A2, then the data segment in *ReadBuffer* started from 72 to the value specified in *ip_totallength* field are copied into *NTcreate_andX* header record. Figure (3.5) illustrated the position of *NTcreate_andX* header in IP header.

To know if the received packet is client request or server response, the *WordCount* field should be checked. (i.e., If *NTcreate_andX. WordCount* field equal 24, then the current packet is client request and should be analyzed, otherwise the current packet is considered as server response and it is omitted).

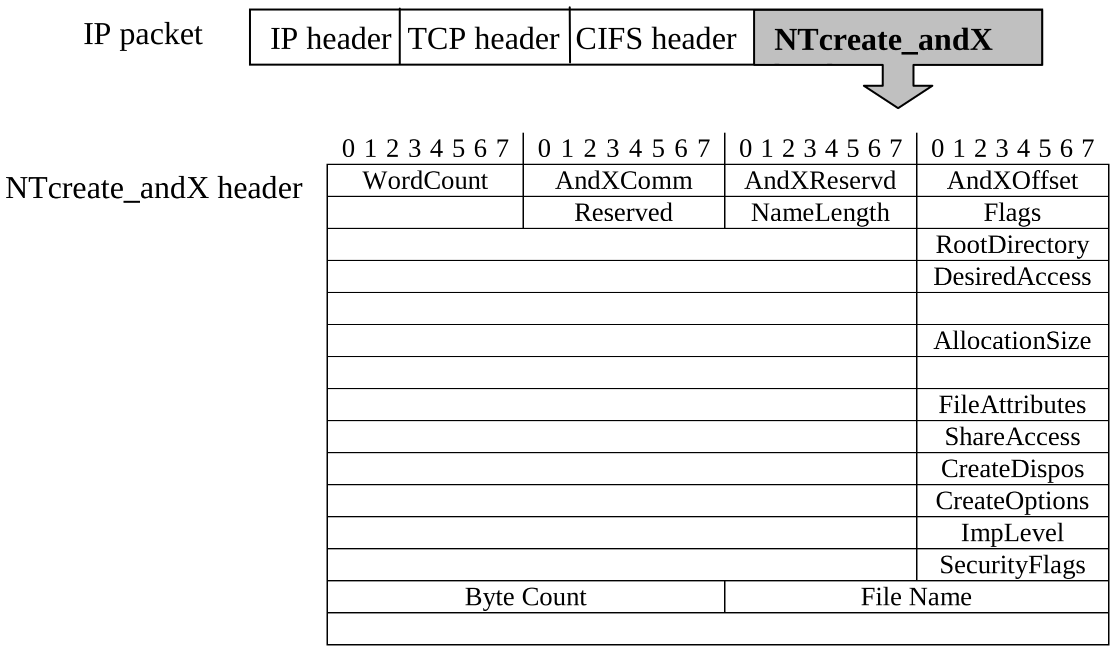


Fig. (3.5) NTcreate_andX header in IP packet [MIC06]

Then, the field that is checked in this step is *NTcreate_andX. DesiredAccess.* The value in this field is used to specify the type of access, and it is represented by access mask encoding.

In *Read* case, the value of *NTcreate_andX.DesiredAccess* field is equal to 0x20000. According to Table (2.2) this value means there is *Read* access to the owner, group, and the discretionary access control list (ACL) of the security descriptor.

The name of file that is accessed by a client and its path is extracted from the received packet. *NTcreate_andX.Filename* field is used to specify the files' names and paths.

2. ***Write Access:*** Like Read access, the type of command packet that is implemented in this access is *NTcreate_andX*. Therefore, this case

applies the same check of client request packet which had been applied in *Read* access.

To check if a client trying to access a shared file for writing, the *NTcreate_andX.DesiredAccess* field must be checked. In *Write* case, the value of *NTcreate_andX.DesiredAccess* field is equal to 0x400000, according to Table (2.2) this value means there is *Write* access to ACL.

3. ***Copy Access:*** If a client trying to access shared files by *Copy* command, then the *NTcreate_andX* packets are issued. Therefore, the same check of client request packet that had been applied in *Read* and *Write* accesses is applied in *Copy* access.

To check if a client copies any file from shared resources, the *NTcreate_andX.CreateOptions* field must be checked. *CreateOptions* field is defined as long type, and its possible values are listed in Table (2.4).

Copy access means that a new file is going to be created. When the user access type is Copy, the *NTcreate_andx.CreateOptions* field must be 0x000002. According to Table (2.4) this value means that a new file is created.

4. ***Delete Access:*** There are two types of command packets could be applied to perform delete access; these types are *CIFS_DeleteDirectory* and *NTcreate_andX*.

CIFS_DeleteDirectory packets are issued when a client try to delete a directory from a shared resource, while *NTcreate_andX* packets are issued when a file is deleted from a shared resource.

The command number of *CIFS_DeleteDirectory* is 1, (i.e., *CIFS.cmd=1*). Figure (2.14) illustrates the header of *CIFS_Delete Directory* command.

In order to copy the remaining data from *ReadBuffer* to the record of *CIFS_DeleteDirectory* header, first the record of *DeleteDirectory* header is created. Table (3.6) shows the *DeleteDirectory* header fields. To check if the received TCP packet is the type *CIFS_DeleteDirectory* or not, *CIFS_cmd* field must be checked. If it is equal to 1, then the block of data in *ReadBuffer* (starting from position 72 up to *ip_totallength*) are copied into *CIFS_DeleteDirectory* header record.

To know the name of the directory that is going to be deleted by a client and its path, the field *CIFS_DeleteDirectory.Directoryname* should be used for this purpose.

In the case of a file deletion from a shared resource by a client, the field *NTcreate_andX.DesiredAccess* must be checked. If *NTcreate_andX. DesiredAccess* field is equal 0x10000, then according to Table (2.2) this value indicate there is delete file access.

Table (3.6) Delete directory header fields

Field	***Size in bytes***
Wordcount	1
Bytecount	1
Bufferformat	1
Directoryname	>=2

The established packet processing module applies the above mentioned checks to allocate the four types of clients' accesses to the shared folders and files. Also, it gets the names of these folders and files and their paths. Code list (3.3) illustrates the implemented steps of the packet processor.

Code List (3.3): Packet Processing Module.

```
Input
        ReadBuffer (0 to 1499) of byte.
        IP header record as shown in Table(3.1).
        CIFS header record as shown in Table (3.4).
Output
        NTcreate_andX header record that shown in Table(3.5)
        DeleteDirectory header record that shown in Table(3.6)
        Access type as string.
        File_name as string.
Begin
Step1: Apply the CIFS header construction function.
Step2: If CIFS.cmd <> 0xA2 then go to step 9.
Step3: Copy data from ReadBuffer (72) to ReadBuffer(ip_totallength) into
        NTcreate_andX header record.
Step4: If NTcreate_andX.wordcount<>24 then go to step13.
Step5: If NTcreate_andX.DesiredAccess=0x20000 then access_type="read" and
        go to step 12.
Step6: If NTcreate_andX.DesiredAccess = 0x400000 then set access_
        type="write" and go to step 12.
Step7: If NTcreate_andX. CreateOptions=0x000002 then set access_ type="copy"
        and go to step12.
Step8: If NTcreate_andX.DesiredAccess=0x10000 then set access_ type="delete"
        and go to step 12, else go to step 13.
Step9: If CIFS.cmd <> 1 then go to step 13.
Step10: Copy data from ReadBuffer (72) to ReadBuffer (ip_totallength) in
        DeleteDirectory header record.
Step11: Set file_name to DeleteDirectory.DirectoryName and go to step 13.
Step12: Set file_name to NTcreate_andX.FileName.
Step 13: End.
```

3.7 Saving Packet Module

After applying the four previous modules (i.e., packet capture, packet analyzer, packet filter, and packet processes) on the received packets, some information from each extracted packet is saved before sent to server machine. This module is used for saving some fields which are extracted from IP, TCP, CIFS, and NTcreate_andX packets and send it to server machine.

Saving and retrieval a data from binary files is faster than other types of files (for example, text files). Therefore, in this work binary files are used to save the extracted information. These files accessed by administrator only (i.e., they couldn't accessed by users). In each run, the data append to these files until the administrator terminates these data. In order to access these binary files by the server machine, they are defined as shared files. Shared file means that they could be accessed at any instance by more than one application. This binary shared file is saved in a directory, which should already be shared on the network to call it by the server machine.

A new array called *Shared* had been created on each client to collect all extracted fields before saving them in a binary shared file. The extracted fields are saved in *Shared* array according to the following steps:

Step1: The two fields (source IP, destination IP address) which are extracted from IP header record, are saved in Shared array as follows:

Shared (i).source_IP= ipheader.ip_srcaddr.

Shared (i).dest_IP= ipheader.ip_destaddr.

Step2: The two fields (source port, destination port) extracted from TCP header record, are saved in Shared array as follows:

Shared(i).source_Port= tcpheader.Src_portno.

Shared(i).dest_Port= tcpheader.Dst_portno.

Step3: The *Command* field extracted from CIFS header (which is saved in *access type* variable when applying packet processing module) is saved as follows:

Shared(i).command=access type.

Step4: File name or Directory name fields are extracted from *NTcreate_andX* or *DeleteDirectory* headers. These fields are already exist in *File_name* variable, the contents of this variable are copied in *Shared* array:

Shared (i).File_name=File_name

Code list (3.4) shows the implemented steps of saving the packet array in a binary share file.

Code List (3.4) Saving Packet Module.

```
Input
    IP header record.
    TCP header record.
    Access_ type as string.
    File_name as string.
Output
   A binary shared file contains the array "Shared" which is a record type of:
        Source_IP as string
        Dest_IP as string
        Source_Port as integer
        Dest_Port as integer
        Command as byte
        File_name as string
        File_path as string At_time as time
        At_date as date
```

```
Begin
Step1: Set i=0.
Step2: From each received bulk of packets
Step2.1: Set Shared(i).Source_IP to ipheader.ip_srcaddr, and set Shared(i).
         Dest_IP to ipheader.ip_destaddr.
Step2.2: Set Shared(i).Source_port to tcpheader.Src_portno, and set Shared(i).
         Dest_Port to tcpheader.Dst_portno.
Step2.3: Call (Get_ FilePath) function to extract the File_ path from File_name.
Step2.4: Set Shared(i).Filename to File_name, set Shared(i).File_path to
         File_path
Step2.5: If access_type="read" then set Shared(i).command to 1,Else if
         access_type="write" then set Shared(i).command to 2, Else if
         access_type="copy" then set Shared(i).command to 3, Else if access_type
         ="delete" then set Shared(i).command to 4.
Step2.6: Get the time of access and save it in Shared(i).At_time, and get the Date
         of access and save it in Shared(i).At_date, increment i by 1.
Step 2.7: If i<1000 then go to step 2.1(for the new coming bulk of packets)
Step2.8: Open binary shared file and save the "Shared" array in this file, and set
         i=0.
Step2.9: Check in stop command is not prompted then go to step 2.1.
Step3: End.
```

3.8 The Monitoring Unit

Network security monitoring system monitors the clients' accesses to the available storage shared resources, which they exist in server machine. Therefore, the implementation of the proposed system was based on a client-server model, and the monitoring unit is implemented on client side.

The combination of the modules (packet capture, packet analyzer, packet filter, packet processing, and saving packet modules) makes the monitoring unit. Code list (3.5) shows the implemented steps to manage the calling of these five modules to conduct the monitoring activity. While Figure (3.6) shows the layout of monitoring unit.

Code list (3.5): Monitoring Unit

```
Input
        Row of data
Output
        Database packets
Begin
Step1: Do while connection is starts.
Step2: Apply Packet Capture module on the received packet.
Step3: Apply Creating CIFS header module on the output of Packet Capture
        module.
Step4: Apply Packet Processing module on the output of Creating CIFS header
        module.
Step5: Apply Saving Packet module on the output of Packet Capture module and
        the output of Packet Processing module.
Step6: Check if connection is not stop then go to step 1.
Step7: End.
```

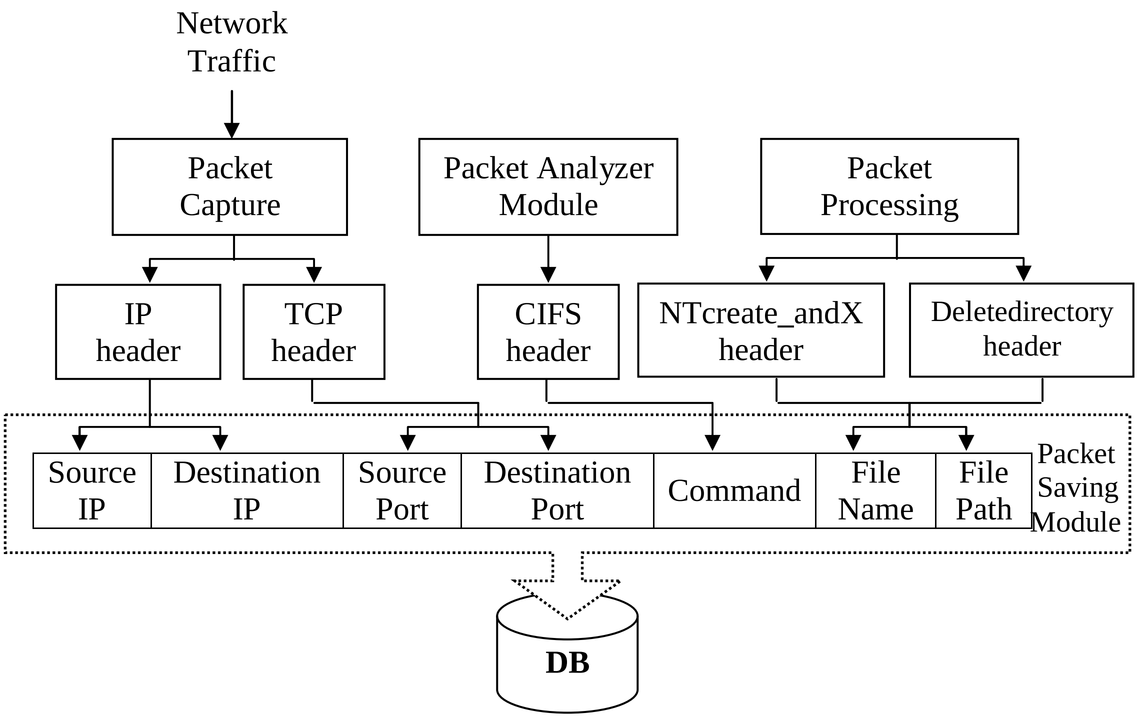


Fig. (3.6) The layout of monitoring unit

3.9 The Proposed Access Control Service

As mentioned in chapter two, the job of Access Control security services is determine which information the user can access and which are not. Therefore, the proposed monitoring unit requires an access control service to control the access privileges of clients.

In order to design a stable access control, the number of folders exist in various PC machines have been calculated. During this check, it is found that maximum number of folders is around to (20000) folders, and for files it is (90000) files on the hard disks of the tested PC machines. Because of this huge numbers of folders and files in some machines, the first two methods of access controls (i.e., protection table, capabilities based) are unsuitable to be implementing in this work. Therefore, the method Access Control Lists (ACLs) is the most suitable one to this work. The proposed access control and ACLs are similar in the way of indicating for each object which a specific user is allowed to access that object. Figure (3.7) shows the general format of the proposed access control.

Folder 1	User 1 RWCD	User 2 RWCD	……	User n RWCD
Folder 2	User 1 RWCD	User 2 RWCD	……	User n RWCD
. .				
Folder i	User 1 RWCD	User 2 RWCD	……	User n RWCD

Fig. (3.7) The general format of the proposed access control

The objects of the access control are machines' folders, not machines' files because of the large numbers of files, which leads to a long search time to specify the access permission for each client.

3.10 The Index Filing Module

Network Security Monitoring System monitors the filing contents of the shared resources of the server's machine; therefore, it requires indexing for all folders and files of the server's machine.

Before indexing all drives, folders, and files of the shared storage area in the server machine, three arrays have been created, these arrays are: *index_drives*, *index_folders*, and *index_files*.

Index_drives is an array of records; each record consists of *drive_no, drive name, first_folder, last_folder, first_file, last_file* fields as shown in Table (3.7). While *index_folders* also is an array of records; the elements of the record are *folder_number, folder_name, folder_path, drive_no* fields as shown in Table (3.8). *Index_files* is an array of records, each record consist of *file_number*, *file_name*, *file_path*, *drive_no* fields, as shown in Table (3.9).

Table (3.7) Index_Drives DB

Field Name	Size (Byte)	Description
Drive_no	1	Specifies the host drive number.
Drive_name	2	Specifies the host drive name.
First_folder	2	Specifies the index number of first child folder saved in this drive.
Last_folder	2	Specifies the index number of last child folder saved in this drive.
First_file	4	Specifies the index number of first file exist in this drive.
Last_file	4	Specifies the index number of last file exist in this drive.

Table (3.8) Index_Folders DB

Field Name	Size (Byte)	Description
Folder_no	2	Specifies the index number of the folder.
Folder_name	50	Specifies the name of the folder.
Folder_path	800	Specifies the complete path of the folder.
Drive_no	1	Specifies number of the host drive of this folder.

Table (3.9) Index_Files DB

Field Name	Size (Byte)	Description
File_no	4	Specifies the index number of the file.
File_name	50	Specifies the name of file.
File_path	800	Specifies the complete path of file.
Drive_no	1	Specifies the index number of the host drive of this file.

The steps of file indexing module are summarized by the following:

1. **Reading Hard Disk's Drives:** This module begins to allocate the hard disk's drives of the server machine, and specify a unique number to each drive, and finally save this number in *index_drives.drive_no* field. Also, this module gets the name of each drive and save it in *index_drives.drive_name* field.

2. **Reading Drives' Folders:** The module reads the first folder that exist in first drive using the API function *FindFirstFile*, this function opens a search handle, searches a folder, and returns the name of the first file exist in that folder. Then, as next step the file indexing module reads all folders, subfolders, and all files exist in the searched drive using the API function *FindNextFile*. This function continues the file search from the previous call to the *FindFirstFile* function.

   The *FileName* is a string variable used by *FindNextFile* function, this variable holds the scanned names of folders and files. The attributes of this variable must be checked to recognize whether the detected entity is a folder or a file. If the attributes of *FileName* refers to folder, then the module assign to this folder a unique index number, and save it in *index_folders.folder_no* field. Also it will save the *FileName* in *index_foldes.folder_name* field, and save the complete path of folder in *index_folders. folder_path* field.

   Also this module save for each scanned drive the index number of the first folder detected, in that drive, in *index_drives.first_folder* field, and save the index number of the last detected folder exist in the same scanned drive in *index_drives.last_folder* field.

3. **Scanning Files in Folders:** The files also scanned by checking the attributes of *FileName*, if the check of folder attributes is failed, then the

scanned entity is considered a file. The module assign to each detected file a unique index number, and save this number in *index_files.File_no* field, and the *FileName* in *index_files. file_name* field, and saves the complete file path in *index_files.file_path* field.

Also this module save the index number of first detected file in *index_drives.first_file* field, and the index number of the last detected file in *index_drives.last_file* field for each drive. When the search handle became null, then it’s closed using the *FindClose* function.

Saving the index of all folders and files in this way is similar to using pointers (i.e., there is a pointer for each drive refer to the position of its folders and files in the lists), while the interaction between these arrays seems like those in tree structure as shown in Figure (3.8).

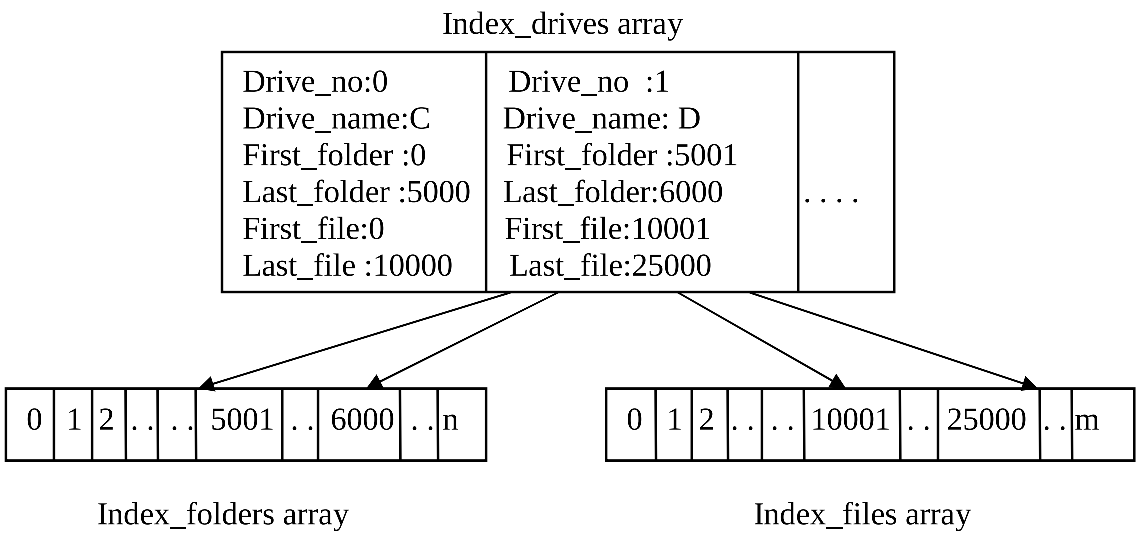


Fig. (3.8) The relationship between the index_drives array, index_folders array, and index_files array

To easily recall the arrays’ information by other modules, the arrays (*index_drives, index_folders,* and *index files)* are saved in three DB table files. In this work, the binary format was used to establish these tables, because this type of files saves and retrieves their data faster than other structural database files (like Microsoft ACCESS). The implemented steps of the module "indexing folders and files" are shown in Code List (3.6).

Some folders which are saved in the same drive have exactly same names. Therefore, the complete path for each folder and file is saved. For example, suppose that the following two paths are found in a host drive, "C:\Document and setting\ computer1 \WINDOWS", and "C:\WINDOWS". In such case, the folder "WINDOWS" exist in two different places in the drive. When the module reading folder's work on this disk, an error ("names duplication") will be happen when these folders are saved in *index_folders* array. For this reason the registration of complete path for each folder and file is necessary.

Code List(3.6): Indexing Files Module.

```
Input
        All hard disk's drives of the server machine.
Output
        Folders DB contains three tables (in binary format), hold the data of the
        following three structure arrays:
Index_drives (0 to 9) array of
        Drive_no as byte
        Drive_name as string * 2
        First_folder as integer
        Last_folder as integer
        First_file as long
        Last_file as long
Index_folders (0 to 20000) array of
        Drive_no as byte
        Folder_no as integer
        Folder_name as string * 50
        Folder_path (0 to 15) as string* 50
Index_files (0 to 100000) array of
        Drive_no as byte
        File_no as long
        File_name as string *50
        File_path (0 to 15) as string*50
Begin
Step1: Read all drives of hard disk by Drive List Box control.
Step2: Set n to the number of drives.
Step3: Set d, fo, fi equal to 0.
Step4: Set d equal to Index_drives(d).drive_no; and set drive1.list(d) equal to
        Index_drives(d) .drive_name.
```

Continue

```
Step5: Set fo equal to Index_drives(d).first_folder; and set fi equal to
       Index_drives(d). first_file.
Step 6: Call FindFirstFile function for the derive (Index_drives(d).drive_name)
       and put its retrieved non-empty string in Filename.
Step 7: Check if Filename=”.” And “..”, then go to step 10.
Step8: Check if the attributes of Filename refers to Folder then
Step8.1: Put fo in Index_folders (fo). folder_no;
Step8.2: Put Filename in Index_folders(fo).folder_name;
Step8.3: Apply get_path function on Filename and put the retrieved path string in
       Index_folders(fo). folder_path;
Step8.4: Increment fo by 1 and go to step 10.
Step9: Put fi in Index_files(fi).file_no; put Filename in Index_files(fi). file_name;
       Apply get_path function on Filename and put the retrieved path string in
       Index_file(fi).file_path; increment fi by 1.
Step 10: Call FindNextFile function and check, if there is another file or folder
       then put the non empty string in Filename and go to step 7.
Step11: Call FindClose function.
Step12: Put fo-1 in Index_drive(d).last_folder, and put fi-1 in Index_drive(d).
       last_file.
Step 13: Increment d by 1; check if d<n then go to step4.
Step 14: Save Index_drives, Index_folders and Index_files arrays in binary files.
Step 15: End.
```

3.11 Tree Builder Module

In this work, the administrator is responsible to define the monitoring rules; he enters to the system after he passes the specific ID and password. He should specify the available access to each user. Therefore, the administration interface shows all listed folders, those registered in folders database, to administrate in a clear form and for easily used. The *Treeview* control is used to view all registered folders for rules assignment.

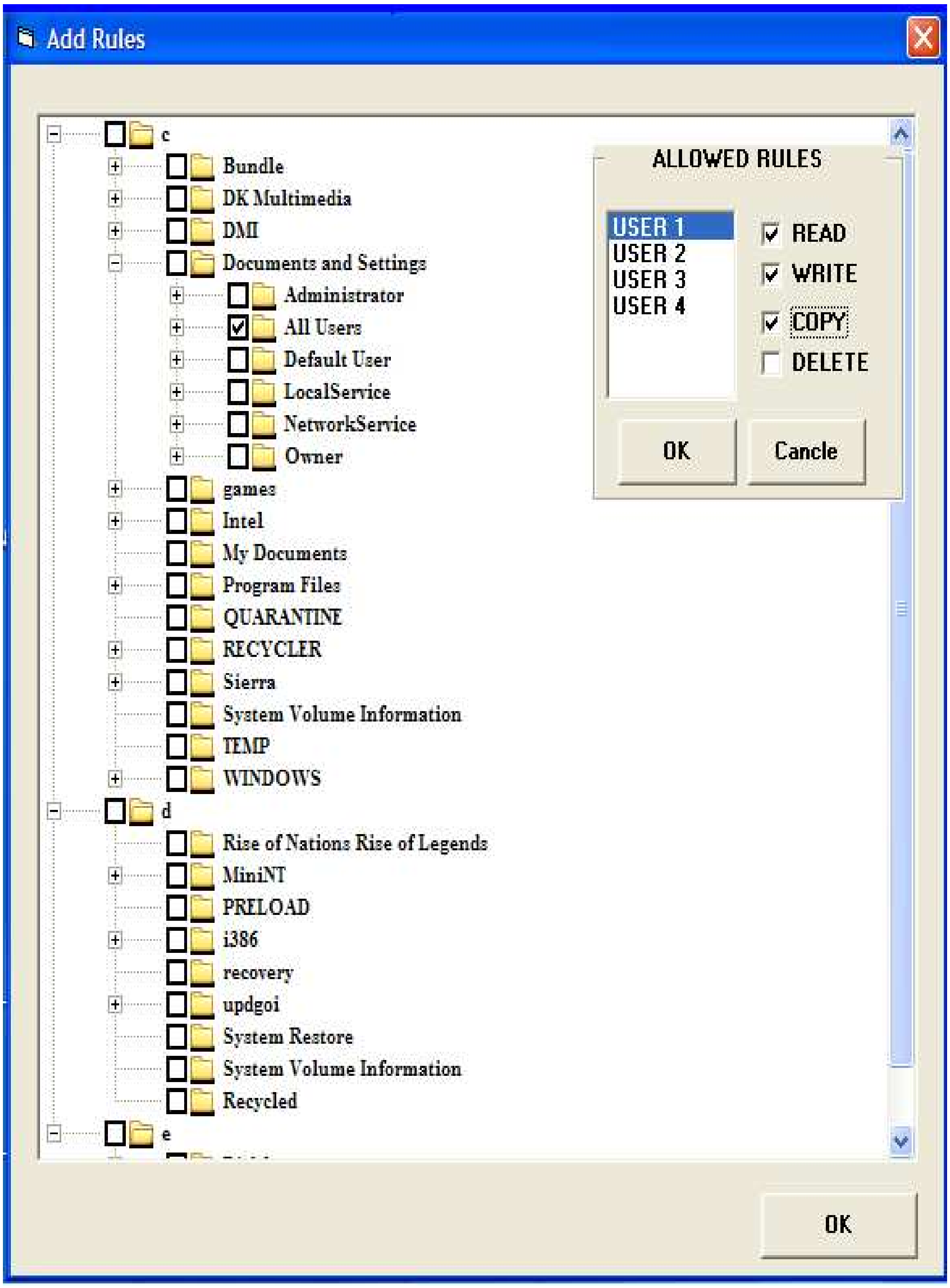
The *TreeView* control is designed to display data which is hierarchical in nature, such as organization trees, the entries in an index, the files and folders on a disk.

A node in this tree can be expanded or collapsed, depending on whether or not it has child nodes or not. The expanded or collapsed nodes events are important to administrate the hierarchal folder list to the desired depth (as he wants).

Also a node can be checked by its check box. This property is very important in rules specification. In the proposed system, folders are checked by administrator, if he check any folder, it means that he want to specify rules to that folder. Immediately, a small window appears for adding rules. Figure (3.9) shows the list of all drives exist on a server machine with their hosted folders using *TreeView* control.

The tree builder module builds the tree by load data from the folders database, code list (3.7) shows the implemented steps of tree builder.

When using tree builder module; it is important to notice that *index_tree.rules* field, which is the output of this module, is still empty, this field is filled when using the “add rules” module.



Add Rules

- ☐ c
  - ☐ Bundle
  - ☐ DK Multimedia
  - ☐ DMI
  - ☐ Documents and Settings
    - ☐ Administrator
    - ☑ All Users
    - ☐ Default User
    - ☐ LocalService
    - ☐ NetworkService
    - ☐ Owner
  - ☐ games
  - ☐ Intel
  - ☐ My Documents
  - ☐ Program Files
  - ☐ QUARANTINE
  - ☐ RECYCLER
  - ☐ Sierra
  - ☐ System Volume Information
  - ☐ TEMP
  - ☐ WINDOWS
- ☐ d
  - ☐ Rise of Nations Rise of Legends
  - ☐ MiniNT
  - ☐ PRELOAD
  - ☐ i386
  - ☐ recovery
  - ☐ updgoi
  - ☐ System Restore
  - ☐ System Volume Information
  - ☐ Recycled
- ☐ e

ALLOWED RULES

USER 1
USER 2
USER 3
USER 4

☑ READ
☑ WRITE
☑ COPY
☐ DELETE

OK Cancle

OK

Fig. (3.9) The Tree View of administrator interface

Code List (3.7) Tree Builder Module.

```
Input
        Folders DB table contains the index_folders array.
        Drives DB table contains the index_drives array.
Output
        A binary file contains the index_Tree(0 to 20000) array of record, each
        element consists of:
            Node as as string * 50
            Index as integer
            Path as string * 750
            Rules (1 to 4) as byte
Begin
Step1: Get index_drives and index_folders arrays from DBs.
Step2: Set d = 0, set index=0, and enable the treeview.checkbox.
Step 3: Check if index_drive(d).drive_name= null then go to step 10.
Step4: Set i = index_drives(d).startfolder.
Step5: Check if i > index_drives(d).endfolder then go to step 9.
Step6: Check if index_folders(i).folder_name and index_folders(i).folder_path is
        exist in the tree then go to step 8.
Step7: Set index_tree (index).node = index_folders(i).folder_name, set index_tree
        (index).index = index, set index_tree (index).path = index_folders(i).
        folder_path, increment index by 1.
Step8: Increment i by 1 and go to step 5.
Step9: Increment d by 1 and go to step 3.
Step10: End.
```

3.12 Add Rules Module

This module is responsible for initiating the rules (including access privileges) using the database table for file indexing (folders database). This module uses the output data of the module “file indexing”.

The module “add rules” requires applying the tree builder module in each run; because the tree is terminated after the module is stopped. After apply tree builder module, the tree of folders appears to administrator in order to initialize the rules. This tree contains check box on the left of each folder, by which he can specify (choose) the folders that can be accessed by specific user, see Figure (3.9). Also this tree contains a small window on the top right corner, by which administrator can assign the allowed rules. Code list (3.8) shows the steps of add rules module.

Code List (3.8) Add Rules Modules.

```
Input
        A binary file contains the index_Tree (0 to 20000) array of record, each
        consist of:
            Node as integer
            Index as string * 50
            Path as string * 750
            rules (1 to 4) as byte
Output
        A binary file contains the index_Tree (0 to 20000) updated by new rules.
Begin
Step1: Apply Tree Builder module on index_Tree array.
Step 2: Check if tree.node.checkbox<>true then go to step 8.
Step 3: Check the list of users, if list_users are checked then set i= user number,
        Else go to step 8.
Step4: Check if rules(i) <= 0 then go to step 6.
Step5: Check if read.check=true then set rules(i)=rules(i) or 1,
        if write.check=true then set rules(i)=rules(i) or 2,
        if copy.check=true  then set rules(i)=rules(i) or 4,
        if delete.check=true  then set rules(i)=rules(i) or 8.
Step6: Increment i by 1 and check if i <=4 then go to step4.
```

Continue

Step 7: *Check if there is more specified rules for such node then go to step 2.*

Step8: *Open a binary file, save rules, and close file.*

Step9: *End.*

3.13 Rules Implementation

In order to reduce the memory consumption, the assigned rules have been registered in bitmap form (i.e., assigning one bit to specify the status of one access type). This registration mechanism led to using an array of bytes, called *Rules*. The length of this array is equal to the number of monitored users' (i.e., the number of users have privilege to use the shared storage resources). If the number of users equal to n, then the length of *Rules* array equal to n.

In this work, four types of folders and files accesses are considered: Read, Write, Copy, and Delete. The status (permitted or not) of each access type needs one bit to be represented; therefore, 4 bits are needed to represent the status of the four types of access. So, in this work, each entry of *Rules* array had given 8 bits (1 byte). The most right 4 bits are used, and the other 4 bits are ignored as shown in Figure (3.10).

Bit number	1	2	3	4	5,6,7,8
Represent	READ	WRITE	COPY	DELETE	Ignore

Fig. (3.10) Bitmap representation

Each element of the array hold the assign rules for certain user, for example *Rules* (1) contains the rules of user number 1, *Rules* (2) contains the rules of user number 2, …etc. To assign rules for each folder for any user, the *Rules* array is presented in form of table as shown in the example shown in Figure (3.11). In this example, four users are included in the rules table. Therefore, the length of *Rules* array equal 4 for each folder or drive. In this example, user1 which is represented by *Rules* (1), can read, write, copy, and delete the folder "C:\WINDOWS", because the value of *Rules* (1) is equal 15. While, user2 which is represented by *Rules* (2), can only read and write on the same folder. The range of values of *Rules* array in the proposed system is [0, 15].

Folder Name	Folder path	"User 1" Rules(1)= 15				"User 2" Rules(2)= 3				"User 3" Rules(3)= 4				"User 4" Rules(4)= 1			
WINDOWS	C:\	D	C	W	R	D	C	W	R	D	C	W	R	D	C	W	R
		1	1	1	1	0	0	1	1	0	1	0	0	0	0	0	1

Fig. (3.11) An example of *Rules* array's representation

3.14 Modify Rules Module

The steps of modify rules module can be summarized as follows:

1. Taking data from *index_tree* array.
2. Applying tree builder module on *index_tree* array.
3. Display the exist rules, and gives the administrator the ability to modify the exist rules.
4. Saves the changes in *index_tree* array.

The administrator can adds or erases any type of access to any folder for any user. Code list (3.9) shows the implemented steps of the module "modify rules". In this code list, building the view tree and fixing the existing rules are illustrated by steps [1-8]. In step 9, the module of add "rules" module is called to add other new rules.

Code List (3.9): Modify Rules Module

```
Input
        A binary file contains the index_Tree (0 to 20000) array of record
        (consist of Node, Index, Path, rules()):
Output
        A binary file that contains index_Tree hold the modified rules.
Begin
Step 1: Apply Tree Builder module on index_Tree array.
Step2: Set i = 0.
Step3: Check if index_tree(i).node= null then go to step 9.
Step4: Check if index_tree(i).node.check<>true then go to step8 .
Step5: Set j=0; check if  index_tree(i).rules(j) <= 0 then go to step 7.
Step 6: If index_tree(i).rules(j) and 1 < >0 then read.check=true,
        if index_tree(i).rules(j) and 2 < >0 then write.check=true,
        if index_tree(i).rules(j) and 4 < >0 then copy.check=true,
        if index_tree(i).rules(j) and 8 < >0 then delete.check=true.
Step7: Increment j by 1, and check if j<=4 go to step 6.
Step 8: Increment i by 1 and go to step3.
Step 9: Call Add rules module beginning from step2.
Step 10: End.
```

3.15 Report Module

Report module is responsible for binding the monitoring unit with administration interface, by taking the output of monitoring unit and the output of administration unit. This module compares between these two outputs, and makes a report about the users accesses (privileges). The implemented steps of this module are show in code list (3.10). In this code list, there is a calling to get_user function in step5, the input of this function is “source IP number” and the output is the user number which is assigned to that IP, the implemented steps of this function shown in code list (3.11).

Code List (3.10) Report Module.

```
Input
  (i) A binary shared file contains the array Shar (0 to 10000) of record, each
    record consists of the following members:
        Source_IP as string
        Dest_IP as string
        Source_Port as integer
        Dest_Port as integer
        Command as byte
        File_name as string
        File_path as string
        At_time as time
        At_date as date
  (ii) A binary file that contains the index_Tree (0 to 20000) array of records,
    each record consists of the following members:
        Node as integer
        Index as string * 50
        Path as string * 750
        rules (1 to 4) as byte
Output
        Report
Begin
Step1: Set i = 0, set j = 0.
Step2: Check if shar(i).source_ip = null then go to step12.
Step3: Check if index_tree(j).node = null then go to step 11 .
Step4: Check if index_tree(j). node doesn't exist in shar(i).file_path then go to
        step 10.
```

Continue

```
Step 5: Call get_user function.
Step 6: Check if index_tree(j).rules(user) > 0 then call get_rules function, Else
        set allowed=false and go to step 8.
Step7: Check if index_tree(j).rules(user)=shar(i).command then set
        allowed=true, Else set allowed=false.
Step8: Add to report the items shar(i).source_ip, shar(i).dest_ip, shar(i).
        source_port, shar(i).dest_port, shar(i).command, shar(i).file_name,
        shar(i).file_path, shar(i).at_time, shar(i).at_date.
Step 9: Check if allowed=true then add to report "allowed access", Else add to
        report "not allowed access".
Step 10: Increment j by 1 and go to step 3.
Step 11: Increment i by 1 and go to step 2.
```

Code list (3.11) Get_User Function

```
Input
        shar(i).source_ip as string.
Output
        User as byte.
Begin
Step1:  If shar(i).source_ip= 10.0.0.1 Then user = 1,
        Else If shar(i).source_ip= 10.0.0.2 Then user = 2,
        Else If shar(i).source_ip= 10.0.0.3 Then user = 3,
        Else If shar(i).source_ip= 10.0.0.4 Then user = 4,
Step2: End.
```

Republic of Iraq
Ministry of Higher Education and Scientific Research
Al-Nahrain University
College of Science

Network Security Monitoring System

A Thesis
Submitted to College of Science of AL-Nahrain University in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

By

Zainab Hyder Ameen AL-Essa

(B.Sc. 1999)

Supervisor

Dr. Loay E. George

March 2008 *Rabia I 1429*

بِسْـمِ اللهِ الرَّحمنِ الرَّحِيـمِ

يُؤتـى الحِكمَةَ مَن يَشَــآءُ وَمَن يُؤتَ
الحِكمَــةَ فَـقَد أُو تِيَ خَـيراً كَثـِـيراً
وَمَا يـَذَّكَّرُ
إلا أُوْلُـواْ الأَلبَـابِ

صَدَقَ الله العَليُّ العَظيم

البقرة ٢٦٩

Acknowledgement

I would like to express my deep gratitude and sincere thanks to my supervisor ***Dr. Loay E. George*** *for guidance, assistance and encouragement during the course of this project.*

Grateful thanks for the Head of Department of Computer Science of Al-Nahrain University ***Dr. Taha S. Bashagha*** *for the continuous support during the period of my studies.*

I owe deep gratefulness to my family: ***my parents****,* ***uncle****, and* ***brothers*** *for their attention and encouragement, to* ***my*** *__husband__ and __children__ for their constant concern and their precious words that gave me strength and power when it was desperately needed, to __my aunt__ and __cousins__ for their prayer.*

Grateful thanks to my group fiends specially __Rasool Husham__ and __Haider Majeed__ for their helpful during the period of this project.

Zainab

Dedicated To……….

My Parents

My Husband

My Brothers

My Children

With My Love

Zainab

Abstract

This research is concerned with the design and implementation of a network security monitoring system. A focus was put on monitoring the shared network resources (specifically filing system). In this work, the layers of TCP/IP suite have been studied and their roles in the process of network monitoring were defined, it was found that the internet layer can play the major role. Also, in this project a filing index system to index the files of network resources was built.

The layout of the established monitoring system is composed of two major units (i.e., monitoring and administration units). The monitoring unit works on client side; it monitors all users' accesses to network resources by capturing the IP packets, and then analyze, filter, and assess their security aspects. Finally, it saves some of extracted parts of the IP packets in a database. The administration unit work on server side, it is used for indexing the network resources (i.e., shared files and folders). This unit permits the administrator to assign some available rules to manage the users' accesses. Finally, some reports could be produced, by merging the outputs of both units (monitoring and administration). The registered information about the captured packets are compared with the assigned access' rules for each subject to produce the periodic reports.

The results of the conducted tests indicate that the stage of filtering out the unnecessary packets is very important. If the monitoring unit considers only the relevant packets, then the performance of the system increases and the performance will be stable even when there is high network traffic load.

The proposed monitoring system has been established using Windows API functions with Microsoft Visual Basic 6.0.

List of Abbreviations

ACL	Access Control List
ARP	Address Resolution Protocol
ATM	Asynchronous Transfer Mode
CIFS	Common Internet File System
DECnet	Digital Equipment Corporation Network
DNS	Domain Name System
FTP	File Transfer Protocol
HTTP	Hyper Text Transfer Protocol
ICMP	Internet Control Message Protocol
ID	Intrusion Detection

IDS	Intrusion Detection System
IP	Internet Protocol
IHL	Internet Header Length
IPX/SPX	Internetwork Packet Exchange/ Sequence Packet Exchange
ISO	International Standard Organization
LAN	Local Area Network
MAN	Metropolitan Area Network
MID	Multiplex ID
Negprot	Negotiate protocol
NetBEUI	NetBIOS Enhanced User Interface
NetBT	NetBIOS over TCP/IP
NetBIOS	Network Basic Input Output System
NFS	Network File System
NSM	Network System Monitor
OSI	Open System Interconnection
PC	Personal Computer
PID	Process ID
RIP	Routing Information Protocol
SMB	Server Message Block
SNMP	Simple Network Management Protocol
Tcon	Tree connect
Telnet	Telecommunication Network
TCP	Transmission Control Protocol
TID	Tree ID
TOS	Type Of Service
UDP	User Datagram Protocol
UID	User ID

WAN	Wide Area Network

Contents

References

[ABE06] Abed, S., S.; **”Low Level Security”**, M.Sc. thesis, University of Technology, 2006.

[ALI03] Ali, Z., A.; **“A Real Time Packet Monitoring For Network Intrusion Detection System”,** M.Sc. thesis, University of Technology, 2003.

[AND80] Anderson, J., P. ; **“Computer Security Threat Monitoring and Surveillance”**, James P. Anderson Company, Fort Washington, 1980.

[BAC00] Bace, R., G.; **”Intrusion Detection”,** Macmillan Technical Publishing, USA, 2000.

[BIN07] Binary Vision, **“TCP port 139”**, http://www.linklogger.com, Visit Date: May 2007.

[BIZ07] BizNet Communications Inc, **“CIFS Services”**, http://www.biz1.net, Visit Date: July 2007.

[COD07] CodeFX Software solutions for applications and appliances, **”CIFS Explained”**, http://www.codefx.com, Visit Date: May 2007.

[COM95] Comer, D., E.; **“Internetworking With TCP/IP “,** Vol. 1, 3rd ed., Prentice-Hall of India, 1995.

[DAN07] Daniel Petri Ltd ,**”What's Port 445 Used For in Windows 2000/XP”,** http://www.PETRI.co.il, Visit Date: May 2007.

[FIS00] Fisch, E., A.; White, G., B.; **” Secure Computers and Networks”,** CRC Press LLC, 2000.

[GAL98] Galvin, S., **“Operating System Concepts”,** addition Wesely, 5th ed., 1998.

[GAR96] Garfinkel, S.; Spafford, E.,H.; **”Practical UNIX and Internet Security**”, 2nd ed., by O’Reilly and Associates, 1996.

[HEL00] Helton, P. G., **“Security in computing”**, Prentice Hall PTR, 2000.

[HER90] Heberlein, L., T.; **“A Network Security Monitor.”**, Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, pp. 269-304, May 1990.

[HUG06] Hughes, E.; **“Parsing Streaming Network Protocols”**, M.Sc. thesis, Carleton University, 2006.

[IBM07] IBM Internet Security Systems, **“Port 139 NetBIOS”**, http://www.iss.net , Visit Date: May 2007.

[JAI97] Jaiswal, S.; **”TCP/IP” ,** by Galgotia Publications, 1st ed., 1997.

[JAV07] Javvin Technology, Inc.; **“Microsoft CIFS”**, http://www.javvin. com/protocolCIFS.html , Visit Date: May 2007.

[MCH00] Mchugh, J.; Christie, A.; Allen, J.; **“Defining your self: The Role of Intrusion Detection Systems”**, IEEE Software, No 5, pp. 42-51, September/ October 2000.

[MIC06] Microsoft Corporation, **“Microsoft Developer Network Library (MSDN)”**, 2006, http://msdn2.microsoft.com

[MOL01] Molina, J.; **“Using Independent Auditors for Intrusion Detection Systems”,** M.Sc. thesis, University of Maryland, 2001.

[MUK94] Mukherjee, B.; Heberlein, T. L.; Todd,L. ; Levitt, K., N.; **“Network Intrusion Detection”,** IEEE Network, Vol. 8, No.3, pp. 26-41, May/ June 1994.

[MUS03] Mustafa, K., H.; **”A Simulated Intrusion Detection System Using Packet Header”**, Ph.D. thesis, University of Technology December 2003.

[PAN05] Pande, B.; Gupta, D.; Sanghi, D.; Jain, S.K.; **“The Network Monitoring Tool-PickPacket”,** Information Technology and Application, Vol. 2, No. 10.1109/ICITA.2005.274, pp. 191-196, July 2005.

[RFC61] Walden, D.; Beranek, B.; **“A Note on Interprocess Communication in a Resource Sharing Computer Network“,** RFC 61, Network Working Group, July 1970.

[RFC791] Postel, J.; **”Internet Protocol ”,** RFC-791, prepared for Defense Advanced Research Projects Agency Information Processing Techniques Office, Information Sciences institute, University of Southern California, September 1981.

[RFC793] Postel, J.; **“Transmission Control Protocol”;** RFC-793, Darpa Internet Program, Protocol Specification, prepared for Defense Advanced Research Projects Agency Information Processing Techniques Office, September 1981.

[RFC1001] Aggarwal, A.; Aguilar, L.; **”Protocol Standard For A NetBIOS Service on A TCP/UDP Transport: Concept and Method”,** RFC-1001, Network Working Group, March 1987.

[RFC1180] Socolofsky, T.; Kale, C.; **”A TCP/IP Tutorial”**, RFC-1180, Network Working Group, Spider Systems Limited, January 1991.

[RFC1244] Holbrook, P.; Reynolds, J. ; **“Site Security Handbook”**, RFC-1244, Network Working Group, July 1991.

[SHA07] Sharpe, R.; **“Just What is SMB?”**, http://samba.anu.edu.au/cifs/docs/what-is-smb.html, Visit Date: May 2007.

[SHE07] Sheif, J., S.; Dearmond, T., G.; **“Intrusion Detection: Systems and Models”**, California Institute of Technology, JPL, Pasadena, California State University, Visit Date: July 2007, http://doi.ieeecomputersocity.org/10.1109/ENABL.2002.1029998,

[TAN03] Tanenbaum, A., S.; **“Computer Networks”**, Prentice-Hall PTR, New Jersy, 4th ed., 2003.

[VID04] Vidstrom, A. ; **“The Use of TCP Port 445 in Windows 2000”**, http://ntsecurity.nu, Visit Date: May 2007.

[VII04] Viipuri, T.; **“Traffic Analysis and Modeling of IP Core Networks”,** M.Sc. thesis, Helsinki University of Technology, 2004.

جمهورية العراق
وزارة التعليم العالي والبحث العلمي
جامعة النهرين
كلية العلوم

نظام المراقبة السريّة للشبكات

رسالة

مقدمة الى كلية العلوم في جامعة النهرين

كجزء من متطلبات نيل شهادة الماجستير في علوم الحاسوب

من قبل

زينب حيدر أمين آل عيسى

(بكالوريوس ١٩٩٩)

المشرف

د. لؤي إدور جورج

آذار ٢٠٠٨ **ربيع الاول 1429**

الخلاصة

يهتم هذا المشروع بتصميم وبناء نظام المراقبة السريّة للشبكات ويركز على مراقبة الموارد المشتركة للشبكة وخاصة ملفات النظام (filing system) . في هذا العمل تم دراسة طبقات مجموعة الـ (TCP/IP) وتم تعريف أدوارها في عملية مراقبة الشبكة، وقد وجد أن طبقة الأنترنيت (internet layer) تستطيع أن تلعب الدور الأساسي في هذه العملية. كما أن هذا المشروع يحتوي على نظام الفهرسة الأضبارية (filing index system) لفهرسة ملفات موارد الشبكة.

يتكون مخطط النظام من وحديتين رئيسين وهما وحدتي المراقبة والأدارة، وحدة المراقبة تعمل على جانب الزبون لتراقب جميع حركات المستخدمين لموارد الشبكة عن طريق ألتقاط حزمة الـ (IP) ثم تحليلها وتصفيتها وتحديد مدى سرّيتها لتخزن البعض من أجزائها المختارة في قاعدة بيانات. أما وحدة الأدارة فإنها تعمل على جانب الخادم لفهرسة موارد الشبكة (أي الملفات والفايلات المشترَكة)، هذه الوحدة تسمح للمدير بتخصيص البعض من القواعد المسموحة لتستخدم في إدارة حركة وصول المستخدمين لموارد الشبكة، ويتم تقديم بعض التقارير عن هذه الحركات بين الحين والآخر عن طريق دمج نواتج هاتين الوحدتين. فالمعلومات التي تم اختيارها وتسجيلها تقارن مع القواعد الموضوعة لتقديم تلك التقارير.

تشير نتائج الأختبار بأن مرحلة تصفية الحزم الغير ضرورية هي مهمة جداً، فإذا أخَذت وحدة المراقبة بعين الاعتبار الحزم المتعلقة بالنظام فقط، فهذا يؤدي الى زيادة كفاءة النظام، كما أن هذه الكفاءة تبقى ثابتة عند زيادة الحمل على الشبكة.

تم تصميم البرنامج باستخدام (Windows API functions) مع لغة (MS Visual Basic 6.0) .