

國立交通大學

資訊科學與工程研究所

碩士論文

於 IP-based 網路利用識別碼進行
封包標記與路徑回溯追蹤之研究與實作

Study and Implementation of Identification-based
Packet Marking and Route Traceback in IP-based Networks

研究生：黃民翰

指導教授：趙禧綠 教授

中華民國九十八年七月

於 IP-based 網路利用識別碼進行
封包標記與路徑回溯追蹤之研究與實作
Study and Implementation of Identification-based Packet Marking
and Route Traceback in IP-based Networks

研究生：黃民翰

Student : Tim Hann Huang

指導教授：趙禧綠

Advisor : Hsi-Lu Chao

國立交通大學
資訊科學與工程研究所
碩士論文



Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年七月

於 IP-based 網路利用識別碼進行封包標記與路徑回溯追蹤之研究與實作

學生：黃民翰

指導教授：趙禧綠

國立交通大學資訊科學與工程研究所碩士班

摘要

隨著網路技術的發展，網路安全的議題逐漸受到重視。許多攻擊者在網路上使用偽裝的來源位址進行封包傳遞，隱藏自己的位置，以致現今路徑回溯追蹤方法，無法找到正確的攻擊來源位址。現今路徑回溯追蹤的方法是利用所收到的封包，取得封包內的來源位址，向來源位址發送封包，所經過的路由器會回傳本身位址資訊，達到路徑回溯追蹤。當來源位址是偽造時，往回發送的封包，傳送的路徑並非攻擊封包使用的傳送路徑，因此追蹤出錯誤的路徑。封包標記是路徑回溯中重要的一環，在封包的傳遞過程中，經過擁有封包標記的主機，受害者收集這些標記的內容，去追蹤出正確的路徑。本篇論文提出一個簡單的封包標記與標記記錄追蹤的方法，在偽裝來源位址的情況下，依然能夠正確追蹤出傳送的來源，對於即時性與非即時性的追蹤都可達成。這個方法需要使用到 IP Option 的欄位，標記方式是將設定的識別碼加入 IP Option 的欄位，並將經過的每一個標記主機記錄下來，最後再使用這些標記資訊，進行路徑的查詢，以找出攻擊者的位置，達到路徑追蹤的效果。

關鍵詞：網路安全、偽造網路位址、封包標記、路徑回溯追蹤

Study and Implementation of Identification-based Packet Marking and Route Traceback in IP-based Networks

Student : Tim Hann Huang

Advisor : Hsi-Lu Chao

Institute of Computer Science and Engineering
National Chiao Tung University

Abstract

Along with the development of Internet, network security becomes important. Many attackers spoofed the source address of the packets in the internet. The method of traceback would not trace the true path of source which is spoofed. The method of traceback used the source address of the packet and sent the packet to the source address. The router along the path will return the IP address of itself. The victim can use these messages to rebuild the path. But the source address is spoofed so that the trace path is wrong. According to this reason, packet marking is used to get the accurate trace path. The packets across the marking machine were marked by marking procedure. The victim could collect or gather the marking information to trace the accurate paths. This paper describes a simple method of packet marking for IP traceback. The packets with spoofing address could be traced the accurate paths by marking information. The Identification-based Packet Marking (IPM) for Real-Time/Non-Real-Time is effective to trace route. The IPM marks identifiers to the IP Option field and loses the marking information. Afterwards, we could find the path of packet's transmission by analyzing the marking information.

Keywords: Network Security, IP Spoofing, Packet Marking, Route Traceback

致 謝

在碩士一路下來，從環境的未知開始熟習，在修習課程結交了許多朋友，共同的努力完成課業上的課題，作業的討論，團隊的合作，從中獲得許多能力，感謝朋友們的幫忙與協助。

在自己的計劃當中，遇到最困難的就是從未碰過的東西，第一次開始接觸 Linux 的時候，就必須朝向最困難的 Kernel 開始修改，新手的自己總是遇到重重困難，藉著網路的搜尋，書籍的借閱以及向相關研究的同學詢問，達成所需要的實作內容，特別感謝同樣研究的同學耐心的教導與回答。

核心編程，算是這全部份最重要也最困難的地方，有了老師指導實作的方向，修改內容符合實際要求，然而常常遇到瓶頸無法解決，花了很多時間在研究程式編寫，然而老師的說法讓我有許多新的想法與思考，從中獲得許多不同的看法，感謝老師在這方面的指示與教導。

在碩士後來的過程中，非常感謝學弟妹的幫助，讓我減輕不少的壓力，能夠順利的準備畢業前夕的資料，也完善的達成最後的目標，變成現在的成果，感謝學弟妹的精力與體力的提供。

最後感謝父母的關心與鼓勵，家庭與外宿有很長一段距離，經常在電話詢問最近的狀況，擔心我身體沒照顧好，或是吃得不夠多，常常寄水果與零食給我，讓我能夠安穩的在這學習與完成自己的學業，謝謝父母們的照顧。

黃民翰 2009 年 7 月 6 日于新竹交通大學

Contents

摘要	iii
Abstract	iv
致謝	v
Contents	vi
List of Tables	viii
Lists of Figures	ix
Chapter 1. Introduction	1
1.1. Traceback Approaches	2
1.2. Contribution	3
1.3. Organization	3
Chapter 2. Related Work	5
2.1. Probabilistic Packet Marking Scheme	5
2.2. Deterministic Packet Marking Scheme	7
2.3. Router Interface Marking Scheme	9
2.4. Logging Scheme	9
Chapter 3. Packet Marking and Route Traceback	10
3.1. Design Conception	10
3.2. The Proposed Packet Marking Module	11
3.3. The Proposed Logging Module	14
3.4. The Proposed Traceback Module	15
3.5. System Architecture	17
3.6. Algorithm Design	18
3.6.1. The Process of Packet Marking	19
3.6.2. The Process of Packet Logging	20
3.6.3. The Process of Traceback	21
Chapter 4. Performance Evaluation	23
4.1. Attack Scenario	23
4.2. Experiment Environment	23
4.2.1. Hardware	23
4.2.2. Network Topology	24
4.2.3. Functions	25
4.3. Experiment Scenarios	26
4.4. Experiment Result	27
4.4.1. Setting Result	28
4.4.2. Sniffer Result	30

4.4.3.	Database Result	31
4.4.4.	Traceback Result	32
Chapter 5.	Conclusion and Future Work.....	35
References	37
Appendix A.	Codes.....	38
Appendix B.	Setting Procedures of an IPM Router	75



List of Tables

Table 1 Fields of IP header.....	12
Table 2 Design of our IP option	12
Table 3 Columns and data type in the table.....	14
Table 4 Hardware specification	24
Table 5 Functions of the machines	26
Table 6 Network setting of all equipment	27



Lists of Figures

Figure 1 Network as seen from a victim of an attack.....	5
Figure 2 Encoding edge fragment into the IP identification field.....	7
Figure 3 DPM only works in the router R ₁	8
Figure 4 Network architecture.....	10
Figure 5 Three modules of IPM router.....	11
Figure 6 Process of traceback scheme.....	16
Figure 7 Example of the traceback scheme.....	17
Figure 8 Overview of System architecture of IPM router.....	17
Figure 9 Components between IPM Router and MCC.....	18
Figure 10 Procedure of Bridge model.....	19
Figure 11 Algorithm of packet marking process.....	19
Figure 12 Procedure of packet transmission through sniffer.....	20
Figure 13 Algorithm of packet logging.....	21
Figure 14 Connection between MCC and IPM router.....	22
Figure 15 Scenario of packet spoofing from an attack.....	23
Figure 16 Network topology.....	25
Figure 17 Traceroute process by victim.....	27
Figure 18 Traceback process by IPM router.....	28
Figure 19 Commands of the IPM router.....	29
Figure 20 Operations of the IPM router.....	29
Figure 21 Domain control and file loading.....	30
Figure 22 Packet information gathered from the sniffer.....	31
Figure 23 Records in the database.....	32
Figure 24 Traceback with the condition “Source IP and Port”.....	33
Figure 25 Traceback with the condition “Destination IP and Date”.....	33
Figure 26 Traceback with the condition “Date and Time”.....	34
Figure 27 Association.....	38
Figure 28 Installation of packets from Internet.....	75
Figure 29 The table “tam” in the database “wnl”.....	76
Figure 30 Commands for bridge setup.....	77
Figure 31 The setting of network rules.....	77
Figure 32 Sniffer for each interface.....	78
Figure 33 Server on the IPM router.....	78

Chapter 1. Introduction

Internet security has been an important research in the recent years. Traceback is one of interesting problems to exhibit the route and pinpoint the true source of received packets. The origins of the problems are the stateless nature of the Internet, the destination-oriented routing and the lack of verification of the source IP address. The attackers utilize these facts to conceal their identities by forging the source address of their attack packets, which is generally known by IP spoofing. IP spoofing technique makes the attackers difficult to detect and trace.

Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks are the threats to the Internet Infrastructure. An Attacker dominates several hosts, called agents, to inundate a large number of packets to the same host, called victim. The enormous volume of traffic aggregates at the victim so that the congestion and packet loss are occurred. Additionally, Resources of the victim are consumed by the traffic of attacks. Hence, The Resources are unavailable for legitimate clients. The quality of the victim is growing worse and being destroyed. DoS and DDoS are the most common to take advantage of IP spoofing. There are two purposes of IP spoofing. One of the purposes is to conceal the identities of attackers so that the victim fails to trace back to the sources of attacks. The other is to make difficult to distinguish the spoofed packets from valid packets. The victims can not verify whether the source address carried by the packet is valid or not. Therefore, it is motive for attackers to force the source IP address.

As already mentioned, IP spoofing is a serious problem. The solution of IP traceback is the major goal. Traceback mechanisms have been proposed to trace the real source of the attackers. The one of the purposes is to cease the attacks at the position nearest to its source in order to reduce waste of network resources. The other is to find the identity of the attackers in order to take other legitimate actions against them. In backtracking techniques, the traceback process is performed hop by hop. It first starts at the victim level. The neighboring network

elements of the victim are transmitted a description of the flow. They are requested to identify if a same flow is routed locally; this supposes that the routed flow is successful to the backtracking mechanism. If so, neighbors should be identified in order to repeat the operation. This operation is performed recursively until identities of attacks are discovered.

1.1. Traceback Approaches

The existing approaches for IP traceback could be grouped into two dimensions: packet marking and packet logging. The main idea behind packet marking is to record path information in packets. Routers write their own identification information into header fields of forwarded packets in mark-based traceback. The victim then retrieves the marking information from received packets and determines the routing path by the collection of marked packets. Due to the limited spaces of the marking fields, routers probabilistically decide to mark packets. Therefore, each marked packet only carries partial information of the path. The path can be constructed by combining the marking information collected from a number of received packets. Such approach is known as probabilistic packet marking (PPM) [1]. PPM incurs little overhead of the packets at routers. However, it requires some of marked packets to construct the path from victim to their origin.

The main idea in packet logging is to record path information at routers. Packets are logged by the routers on the path toward the destination in log-based IP traceback. The path is constructed based on the information of logs at the routers. The log-based approach is more powerful as it can trace attacks that use a single packet. However, it is impractical due to that the enormous resources for packet logs. A hybrid IP traceback approach based on both packet marking and packet logging maintains single packet traceback and alleviates the resources for packet logs. Such approach is the main goal for us to trace the position nearest to the attacks.

Combining two approaches is closed related to the techniques of packet marking and packet logging. Many researchers proposed packet marking approaches for constructing the path toward to the sources of attacks. Majority of them utilizes the fields used unusually in the IP header in order to append the marking information of the network path. Such method does

not increase the traffic load of the internet and does not change the measure of packet proposition. However, it induces errors occurred in the fragmentation and reassembly of IP datagram and is limited marking fields to a small amount of spaces. Others are appends IP Option fields into the packet of IP header in order to attach the marking information to the packets. Compared to the preceding method, the marking fields are possessed of large spaces for appending marking information. Nevertheless, the method raises the traffic load of the internet and increases the measure of each transmitted packets. Hence, both of two methods have distinct consideration. Choosing the suitable method has a great influence on our approach.

1.2. Contribution

In this thesis, the proposed scheme is a hybrid IP traceback approach based on packet marking and packet logging for revealing the attackers at the point nearest to its sources. The packets passed through the marking machines are marked with marking information during the transmission first of all. Furthermore, each marking machines logs the information of marking and the messages related to transmitted packets and stores into databases installed at each marking machine. According to the information of the databases, traceback scheme reconstructs the routing path of the attacks and reveals the marking machines passed through by the packets of attacks.

Due to the proposed scheme, two purpose of IP spoofing will be destroyed. The reason why we can prevent attackers from concealing the identities is that the marking information presents the position of the sources. The spoofed packets can be recognized by filter according to the same marking signs of the received packets. Hence, the spoofed packets transmitted from attackers are marked by marking machines and are identified if the same flow is routed locally by marking information.

1.3. Organization

This thesis is organized as follows. The related work of packet marking, packet logging, and approaches to defend spoofed packets is introduced in Chapter 2. In Chapter 3, the

proposed scheme is studied and properties are discussed. Afterward the algorithms of the marking and logging and the implementation of proposed scheme are showed in Chapter 4. Finally, the thesis concludes with Chapter 5 and future work is presented.



Chapter 2. Related Work

Many researchers proposed mark-based approaches for constructing the path toward to the sources of attacks. The similar approaches are grouped into the same categories by marking techniques. The following session describes several IP traceback approaches to identify attack origin. Figure 1 depicts the network as seen from a victim V. Routers are represented by R_i and an attacker A has an attack to the victim. The node A on the network is the attack origin and the attack path from A is the unique ordered list of routers between A and V. An attack from A to V must traverse the path $R_1, R_4,$ and R_6 .

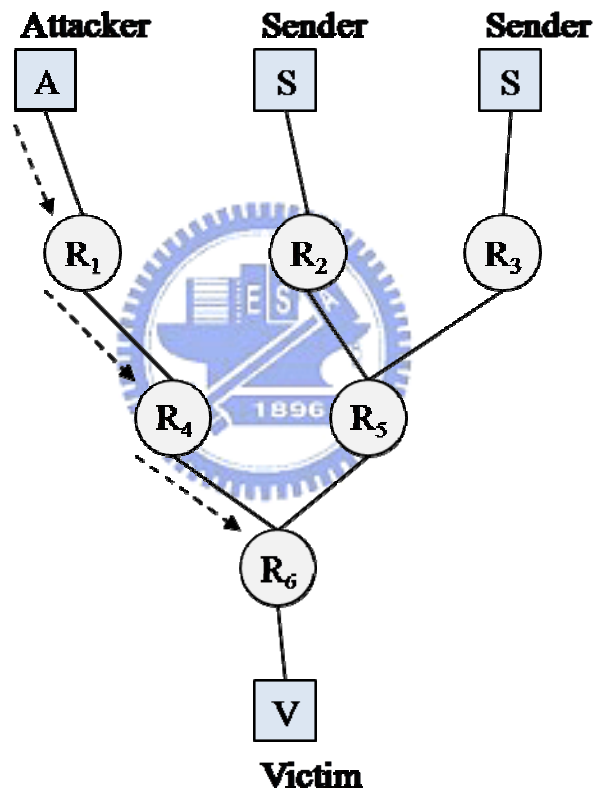


Figure 1 Network as seen from a victim of an attack

2.1. Probabilistic Packet Marking Scheme

An intriguing alternative solution to IP traceback problem is probabilistic packet marking (PPM). The concept of the PPM is marking the packets probabilistically and collecting the information of marking at the victim in order to reconstruct the routing path between the attacker and the victim. The PPM has a series of marking algorithms to

implement the marking process.

The first algorithm, called PPM - Node Append, is the basic idea and the simplest algorithm. The concept is similar to the IP Record Route Option [2]. The algorithm is to append each node's address to the IP option field of the packet as it travels through the network from attacker to victim. Therefore, every packets received by the victim reaches with a complete ordered list of the nodes it traversed. However, it is impossible to ensure that there is sufficient unused space in the packets for the whole list of nodes. Moreover, the attackers can append data into IP option field so that there is not sufficient space for router to append the address of them.

The second algorithm, called PPM – Node Sampling, is to sample one node along the path at a time instead of the entire path in order to reduce router overhead and solve the problem of spaces. The implement of the algorithm requires addition of a write and checksum update to the forward path. A “node” field is reserved in the packet header to hold a single router address. Each router received the packet chooses to write its address in the node field with probability p . The probability of receiving a marked packet from a router d hops away is $p(1-p)^{d-1}$. The victim will have received at least one sample for every router in the attack path after enough packets had been sent by the attacker. Hence, the victim ranks each router by the number of sample and produces the correct attack path. However, the two serious problem of the algorithm are the slow process of reconstructing the path and the confusion of the multiple attackers. A large number of packets sent from attacker are needed for reconstructing resulted in the first problem. The second problem is caused by different attackers exists at the same distance. This algorithm is not robust for multiple attackers.

The third algorithm, called PPM – Edge Sampling, is to explicitly encode edges in the attack path. Two static address-sized fields and a small field are reserved to hold the addresses and the distance. Two fields, called *start* and *end*, are to represent the routers at each end of a link. A small field, called *distance*, is written by the router along the routing path. When a router decides to mark the packet, it writes its own address into to start field and write a zero

into the distance field. Otherwise, if the distance is already zero, it means that the previous router marked the packet. In this case, the router writes its own address into end field and increases the distance field to one. If the router does not decide to mark the packet, it always adds one to distance field. The victim used the edges sampled in the packets to create the graph conducting to the source. Finally, the version of edge sampling is modified so that the requirement of space is reduced by dividing the edges and fragmenting the sampling. After the fragment of sampling, the identification of IP header is used without increasing overhead of the routers. Figure 2 depicts the fragment of encoding edge. The identification is divided into three parts, called offset, distance and edge fragment. The offset field represents the number of the edge fragment. The distance field represents the hops of the router. The edge fragment represents the part of edge fragment. However, such process needs more packets sent by attackers to construct the accurate path. The attacker can inject a packet, which is marked with erroneous information. Such behavior is called mark spoofing. The PPM cannot prevent the packet with mark spoofing from attackers.

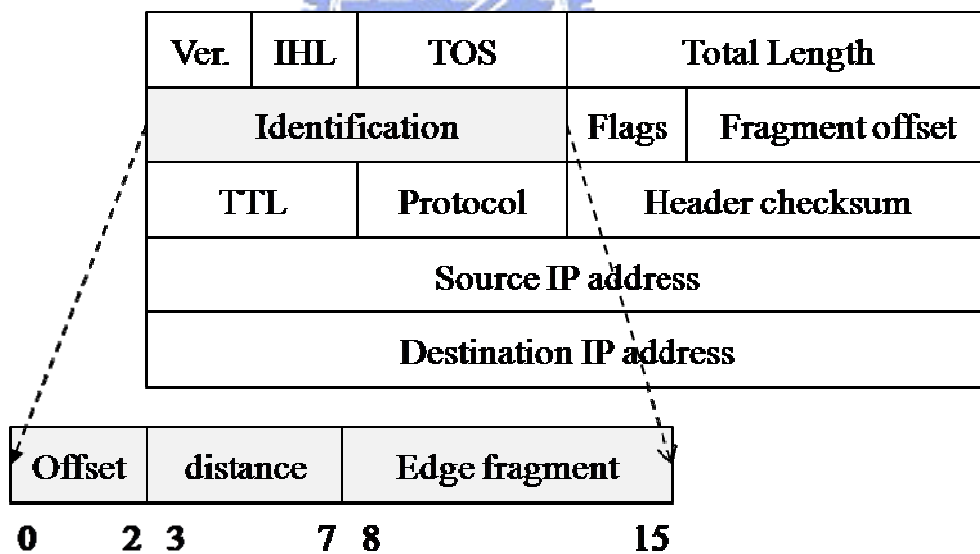


Figure 2 Encoding edge fragment into the IP identification field

2.2. Deterministic Packet Marking Scheme

The opposite of constructing the accurate path, deterministic packet marking (DPM) [3] marks the packets closest to the source. The 16-bit Packet ID field and the reserved 1-bit flag in the IP header are used to mark packets. When a packet passes through the nearest router to

the source, the packet is marked with a part of source IP address. In this case, only address closest to the attacker on the edge routers will participate in packet marking. A 32-bit IP address needs to be transmitted to the victim. This means that a single packet cannot carry the whole IP address in the available 17 bits. An IP address is split into two parts, each of them is 16 bits. The reserved bit is set with the probability p . If the reserved bit is set to 0, the ID field of IP address is the first part. Otherwise, the reserved bit is set to 1 and the ID field of IP address is the second part. Figure 3 depicts router R_1 as DPM router.

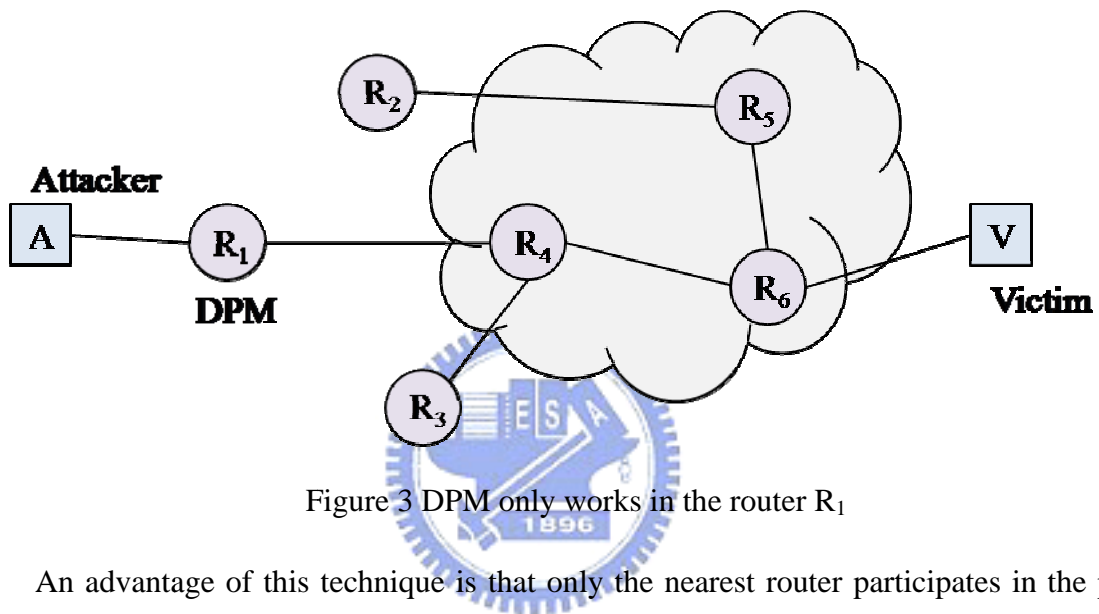


Figure 3 DPM only works in the router R_1

An advantage of this technique is that only the nearest router participates in the packet mark, moreover, it reduces the traffic load of packet marking. However, the packets used the same source IP address by multiple attackers will make the victim confusing with the simples. Due to the drawback, DPM-with address digest [4] modified the DPM and adds an additional function to distinguish the attackers by mark simples.

DPM-with address digest uses the concept of DPM and utilizes a hash function to distinguish the attackers. The different from DPM is using three fields, called Address fragment, Hash digest and Index. The original IP address is divided into more parts in order to write three fields. The packets passed through the same router are containing the same hash identity so that the victim can recognize the different attack path. It extends the advantage of DPM and differs from multiple attackers.

2.3. Router Interface Marking Scheme

The concept of Router Interface Marking (RIM) [5] comes from PPM. The algorithm of RIM is that each routers mark the packets with the probability p . The action of the mark is writes the interface of the router and the hop number into IP header. The ID fields in IP header is used by RIM so that the interface and hop can write to the packet. The ID field is separated into three fields, called IID, XOR and HOP. If the probability p is smaller than a constant, RIM-enable routers write their own identity into the IID field and XOR field, and writes zero into HOP field. Otherwise, routers write their own IID and executes exclusive OR into XOR field, and increment operations on HOP field. The victim collects the marks of the packets received from attack and builds a table in order to compute the same mark of attack path. An advantage of using RIM is that it does not increase traffic load and builds the graph of the attacks. However, it needs to collect the whole marking packets of each RIM-enable router, or the attack path cannot be reconstructed.

2.4. Logging Scheme

There are many challenges to logging. The first one is that the path reconstruction is difficult because of the packets transformed through the network. The next one is that full packet storage is problematic. Memory requirement are unlimited at high line speeds so that the storage never enough. Third one is that traffic repositories may aid eavesdroppers. It may be a privacy risk. The source path isolation engine (SPIE) [6] uses auditing techniques to support the traceback of individual packets while reducing the storage requirements by several orders of magnitude over log-based techniques. The SPIE computes packet digests by invariable fields of IP header and first 8 bytes of the payload so that using packet digests to recognize the different routing packets. It pays to keep an eye on the work of logging. It is important to store up the log under the limited storage.

Chapter 3. Packet Marking and Route Traceback

In this section, we describe our proposed solution which uses packet marking and packet logging techniques to traceback routing path of attack packets. The method of Identification-based Packet Marking (IPM) is marking packets and records marking information into logs of local database at each IPM router. Packets are marked at each IPM enable router in order to trace single packet from the attackers. Therefore, the traceback scheme is using the logs so that non-real-time and real-time traceback schemes are effective.

3.1. Design Conception

Our goal is to design an approach which can trace the closest position where nears attackers. We have to consider legacy routers in the network. Legacy routers should not be replaced by the routers contained the function of marking scheme due to plenty of funds. The approach has to maintain the original network architecture and appends the marking routers to the network architecture. Figure 4 depicts the network architecture in the current network.

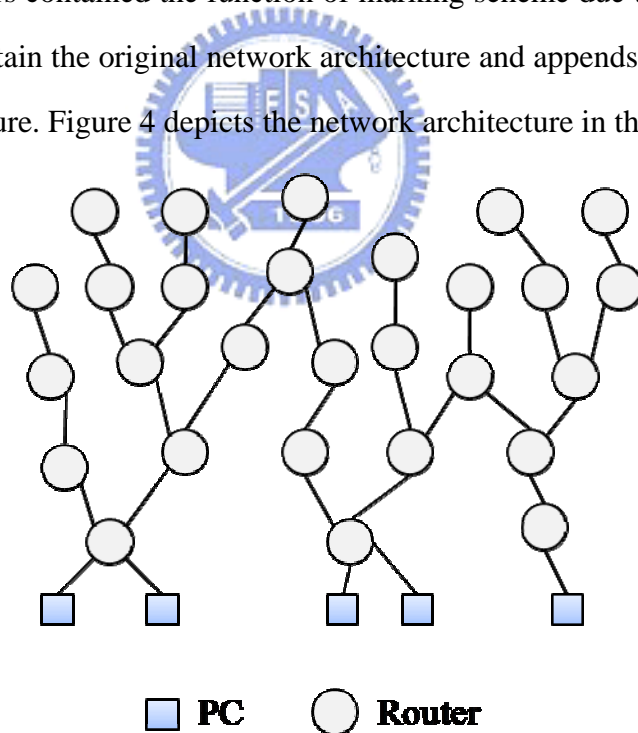


Figure 4 Network architecture

The design of marking scheme has to take legacy routers into account when new routers are placed into the network. The marking technique has to do the marking process and passes through legacy routers. This means that the proposed marking scheme does not work at all

routers and new routers may not be the neighbors. Therefore, IPM routers are placed nearest to the gateway of the local network. Each IPM router contains marking process and local database in order to trace the path. They write marking information into IP header of the packets passed through and record marking information into local databases in the interface of the ingress. The traceback process collects all information from databases of each IPM router and reconstructs the attack path.

Three modules are designed to implement IPM router. Figure 5 depicts three modules in IPM router. Packet marking module marks packets and forwarding them to network. Logging module records packet information Traceback module trace the routing path from database.

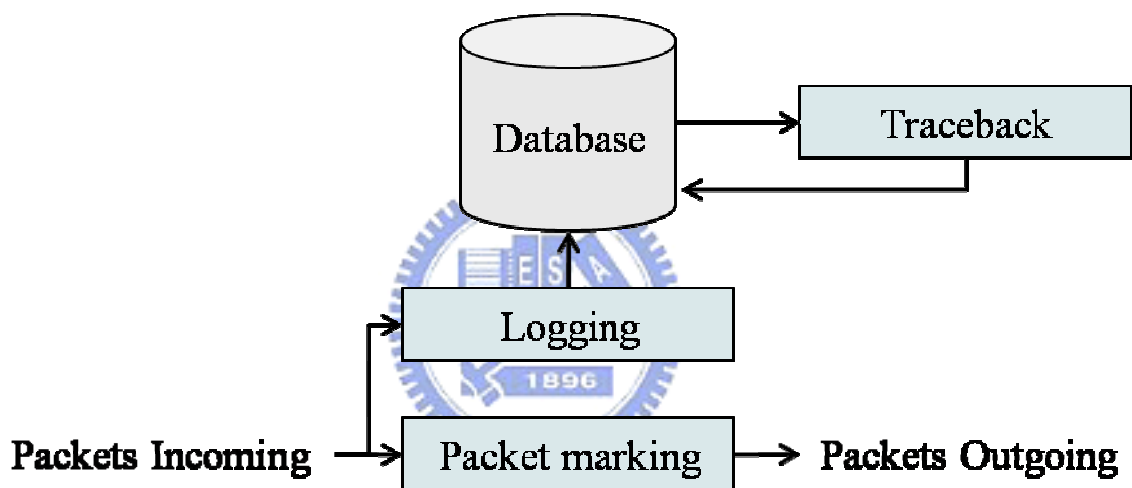


Figure 5 Three modules of IPM router

3.2. The Proposed Packet Marking Module

The design of packet marking utilizes the IP protocol in the existing protocol. The fields of IP protocol are presented in Table 1. The first 20 bytes of IP header are essential for IP protocol during the transmission. When routers or PCs need to send special control message, the IP Option is utilized to contain the messages needed by them. The padding is needed as a filter to guarantee that the data starts on a 32 bit boundary when IP Option field is not a multiple of 32 bits.

Table 1 Fields of IP header

Version	IHL	TOS	Total length	
Identification			Flags	Fragment offset
TTL		Protocol	Header checksum	
Source IP address				
Destination IP address				
IP Option and padding				

We design our own Option for IP Option field for packet marking. There are seven fields in our Option. Table 2 depicts the fields of our option. The following is the description of our design.

Table 2 Design of our IP option

Version	IHL	TOS	Total length	
Identification			Flags	Fragment offset
TTL		Protocol	Header checksum	
Source IP address				
Destination IP address				
Options		Length	IID1	IID2
	IID3		IID4	Hash

The options field is exactly one octet which is their type field, followed by a one octet length field. It is sub-divided into a one bit copied field, a two bit class field, and a five bit option number. These taken together form an eight bit value for the option type field. IP option are commonly refers to by this value. The copied field indicates if the option is to be copied into all fragments. The class field is used for differentiating the group, such as control, debug and measurement. The option number is used for separating different option designs.

The length field is utilized to provide the total length of this option. The different number options have different length so that we can jump into other options by this field. If the

reference is large than the value of this field, it can indicate the wrong message.

Each of the four IID fields has 9 bits. The IID1 and IID2 are utilized to record identities of the first two IPM routers, and the IID3 and IID4 are utilized to record the identities of the last two IPM routers.

The hash field is 12 bytes for verifying the option fields in order to prevent an attacker from manufacturing the option field. We use md5 algorithm to generate the value of the hash field.

Each IPM router has their identities and marking process. The identity of a IPM router is defined by ourselves. The value of the identity is from 1 to 511 because of the size of IID field. The value zero is reserved to indicate if the IID field is used. When the IPM router executes marking process, it first identifies if the option field is existence or not. If so, it check if the hash field is as same as the value it computes. If the computation is correct, it identifies if the IID fields contained the value with zero from IID1 to IID4 in order to write its identity into the fields. While the whole IID fields are not zero, the IPM router copies values from IID4 field into IID3 field and write its identity to IID4. If the hash value is not correct, the IPM router deletes the option of the IP header. If the option field is not existence, the IPM router appends the IP option into the end of IP header so that the marking process can initial the value of option and length and write the identity into IID1. Nevertheless, if the total length is larger than 1492, IPM router does not appends the IP option to IP header because the Maximum Transmission Unit of Ethernet is 1500.

IPM routers append IP option to the packet lead to the different size of the packet. The IHL field specifies the length of the IP header in 32 bit words. The IP option increases the size of the packet so that the value of IHL field has to plus two. The total length contains the length of the datagram so that it needs to plus 8. The changes of IP header occurs header checksum error so that the neighbor router received the packet drops the packet. The IPM router needs to compute the checksum and replace the origin value so that the packet can transmit correctly.

Consequently, IPM routers are not only appending the option to IP header or modifying the value in the option, but also modifying the three values of IP header. Each packet records four identities of IPM routers so that it only contains partial information of IPM routers. The marking scheme marks the packets and forwards the packets to the next router.

3.3. The Proposed Logging Module

We use a sniffer which is a piece of software that grabs all of the traffic flowing into and out of a computer attached to a network. The sniffer can be utilized to gather data necessary for our logging scheme. The IPM router knows that packets received contain the IP option due to the value of IHL. If IHL is larger than 5, the IPM router identifies if the option number equals the number defined by us. If so, it checks on the hash number. If the hash number is correct, it gathers the information from the packet with marking messages before the marking scheme executes.

The logging scheme uses the database to record the information gathered from the packets with marking samples. Each IPM router contains a database. The data needed for traceback are the time, source IP address, destination IP address, protocol type, destination port and IIDs. We construct a table for logging, and the columns are defined in Table 3.

Table 3 Columns and data type in the table

Column	Data Type	Column	Data Type
STime	DATETIME	IIDNUM	SAMLLINT UNSIGNED
ETime	DATETIME	IID1	SAMLLINT UNSIGNED
SIP	INT UNSIGNED	IID2	SAMLLINT UNSIGNED
DIP	INT UNSIGNED	IID3	SAMLLINT UNSIGNED
Protocol	SMALLINT UNSIGNED	IID4	SAMLLINT UNSIGNED
DPORT	SAMLLINT UNSIGNED	IID5	SAMLLINT UNSIGNED

The STime column is the start time of the flow passed through. The ETime column is the end time of the flow passed through. The SIP and DIP column are the IP address of the source

and the destination. The Protocol column is 6 for transmission control protocol (TCP) or 17 for user datagram protocol (UDP). The IIDNUM column is the number of IIDs contained in the IP header. The IID1 to IID5 columns are the identities of IPM routers.

In order to prevent the same records presented in the database from gathering the same flow. The process of logging first gathers the different flow into a buffer which has a fixed size. If the packet of the same flow contains the same information in the buffer, it gathers to the same record so that the same flow only has one record for it. The outdated record stored in the buffer is moved to database when the time of the record is termination or the buffer has been full. The database only contains logs between seven days in order to prevent the full of the storage from logging.

3.4. The Proposed Traceback Module

The concept of traceback scheme is that gathering marking information from each of IPM routers so that all attack paths can be reconstructed by these logs. Each of IPM routers only contains partial information of the attack path. Therefore, we utilize a main control center (MCC) to execute the traceback scheme.

MCC has a graphical user interface (GUI) for users to input the searching conditions. The searching conditions contains the time, source IP address, destination IP address, protocol and destination port. When MCC operates the execution of traceback, it sends a packet with traceback command to the IPM router chosen by MCC. The IPM router starts to search the data satisfied the conditions from the database. Then it sends a packet contained the data back to MCC so that MCC can present the partial path of the attack path. Figure 6 depicts the process of the traceback scheme.

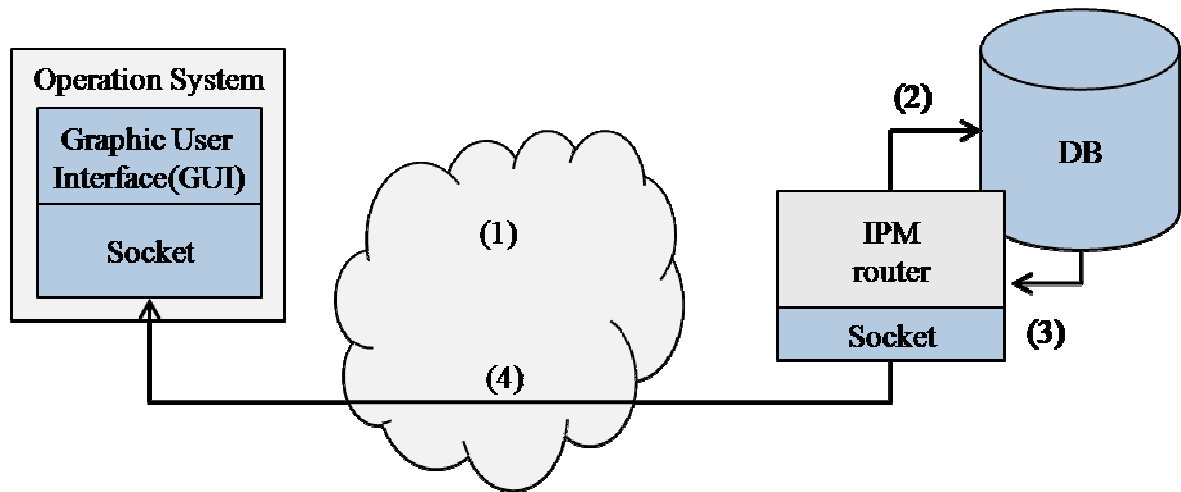


Figure 6 Process of traceback scheme

The traceback scheme traces a partial attack path from an IPM router. The all IPM routers have to cooperate to gather the whole attack path. Figure 7 depicts the example of the traceback scheme. MCC first sends a packet to IPM router #9 so that the MCC receives the packet contained the log of the attack path. MCC can know that the IPM router before IPM router #9 is IPM router #7 by the log of IPM router #9. Then MCC sends another packet to the IPM router #7 to require the log of the attack path. Such process is repeated until the last IPM router #4 is reached. MCC can construct the attack path by gathering the logs of each IPM router. Therefore, MCC finds that the attack path is #4-#2-#6-#3-#7-#9.

However, the process of searching the entire IPM router is not automatic so that it is inconvenient for constructing the accurate attack path. The process done by us is the transmission between the MCC and IPM router. Hence, the transmission between IPM routers may be cooperated to let the process automatically.

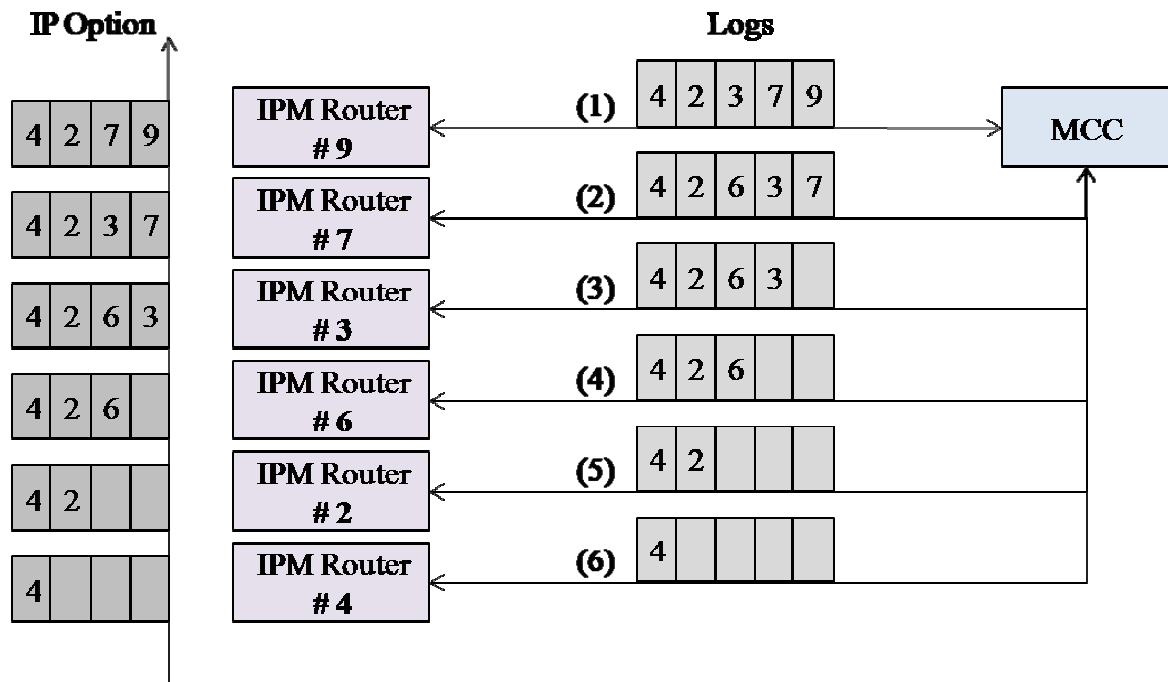


Figure 7 Example of the traceback scheme

3.5. System Architecture

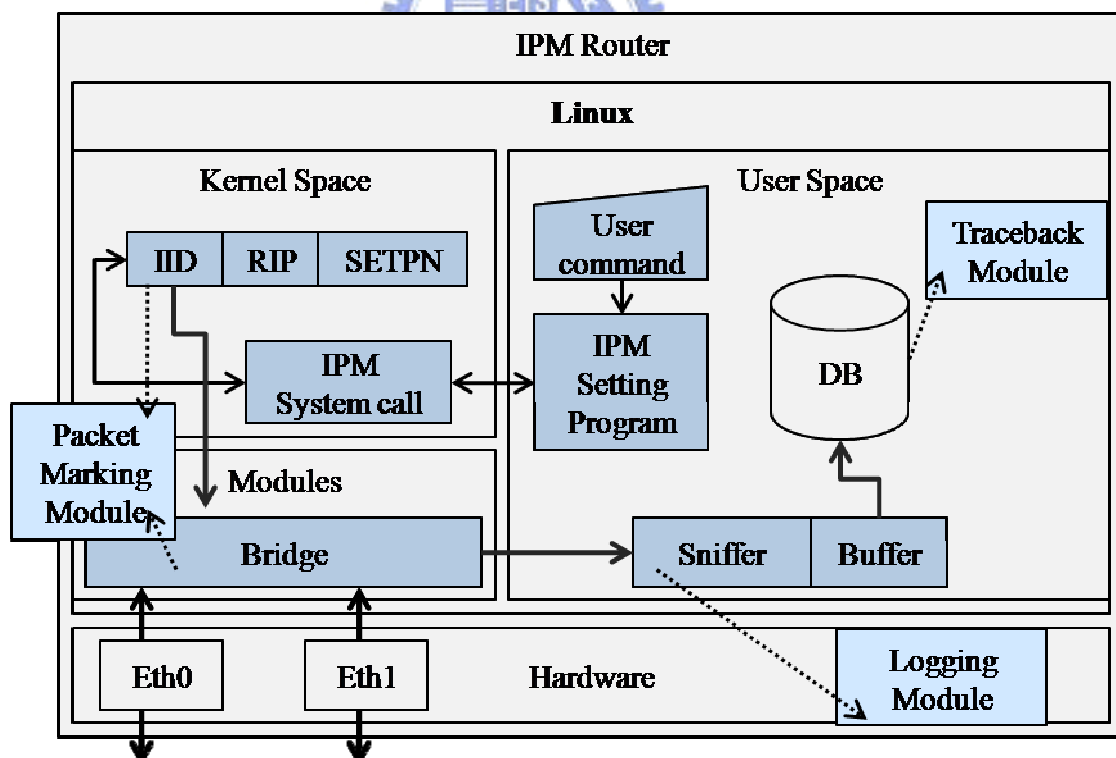


Figure 8 Overview of System architecture of IPM router

The complete components of an IPM Router are shown in Figure 8. The operation

system of IPM router is Linux so that we can modify packets during the forwarding process. In kernel space, we add three variables and system call function into kernel and install the module of the Bridge. Therefore, we install databases for packet logging and write control programs for controlling logs of the database in the user space. The sniffer is utilized to monitor packets transferred between Ethernet interface Eth0 and Eth1. Some programs are designed to modify the variables in the kernel space by system call functions.

The components of the transmission between IPM router and MCC are shown in Figure 9. MCC has the GUI program for user to input the option value and listens to the port number 4862. The IPM router has a program for searching the data from database and listens to the port number 4862.

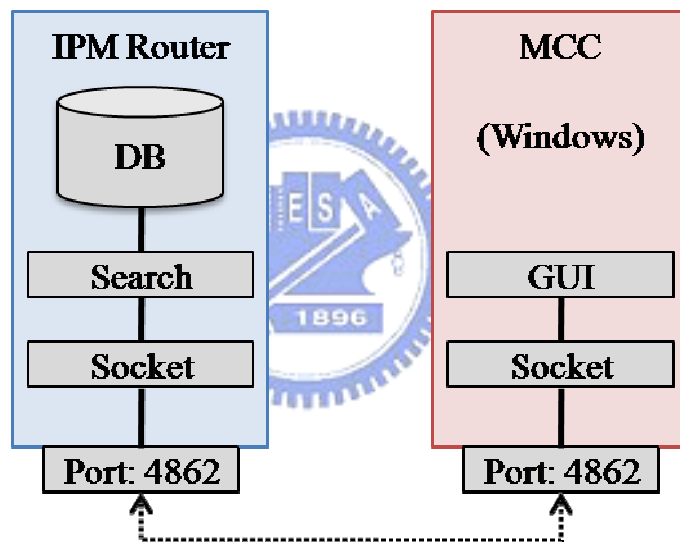


Figure 9 Components between IPM Router and MCC

3.6. Algorithm Design

The algorithm design is sub-divided into three parts. First, the process is executing between the network interface and data-link layer. The packets passed through mark the marking information during this process. Second, the process listened to the interface captures information of the packets passed through the interface. The last one, the process between IPM Router and MCC is the transmission protocol.

3.6.1. The Process of Packet Marking

The packets came from incoming interface pass through Sniffer and Bridge model and forward to outgoing interface. The Bridge model has three parts of the execution. The marking process is written into Forward part so that the packet can combine the marking information. Figure 10 depicts the complete process of the transmission during IPM router.

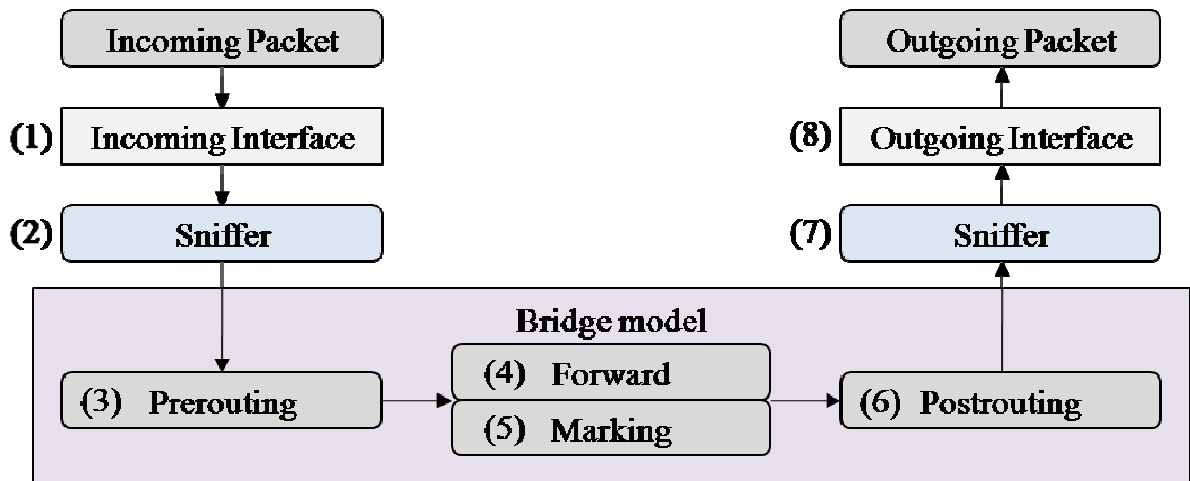


Figure 10 Procedure of Bridge model

The algorithm of packet marking process is shown in Figure 11.

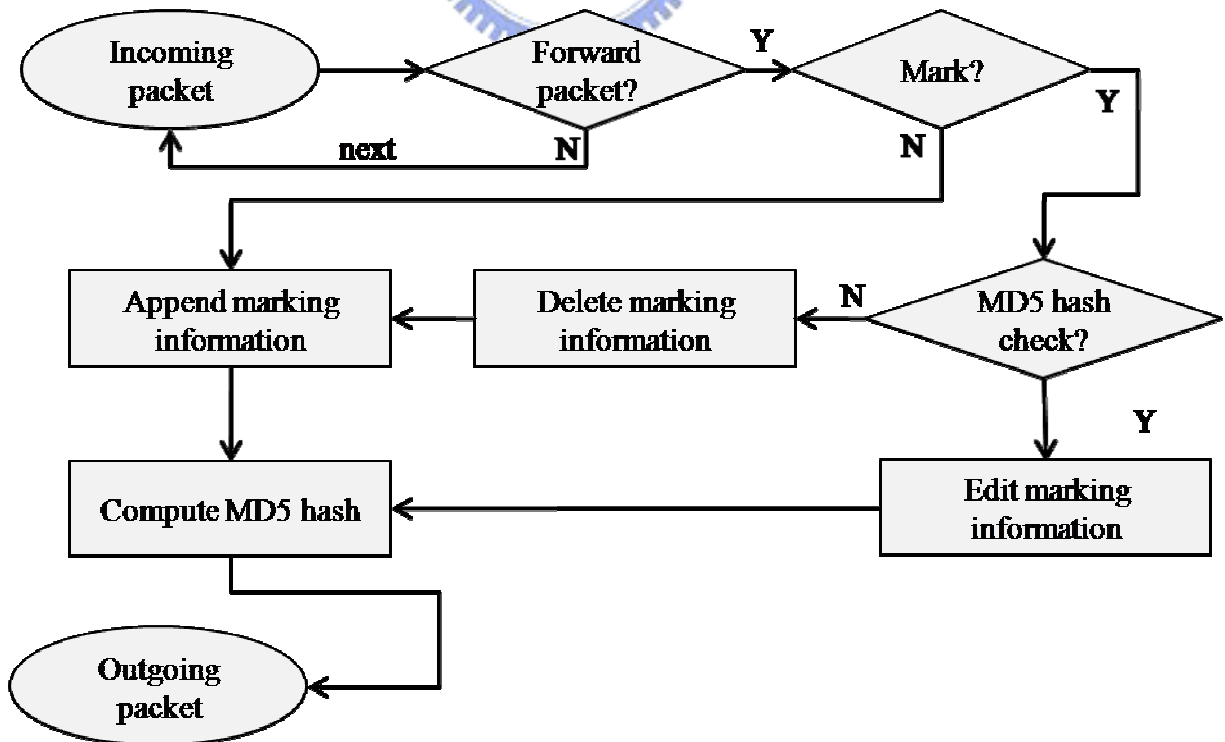


Figure 11 Algorithm of packet marking process

3.6.2. The Process of Packet Logging

The logging process contains the process of the Sniffer and Buffer. The Sniffer can capture and analyze any traffic that pass through the incoming interface. The packets contained the marking information are captured by Sniffer. The marking information is analyzed and written into Buffer for a while. The same marking information in the Buffer is combined into the same record. The records are written to Database while the Buffer is full. Figure 12 depicts the process of the Sniffer and Database.

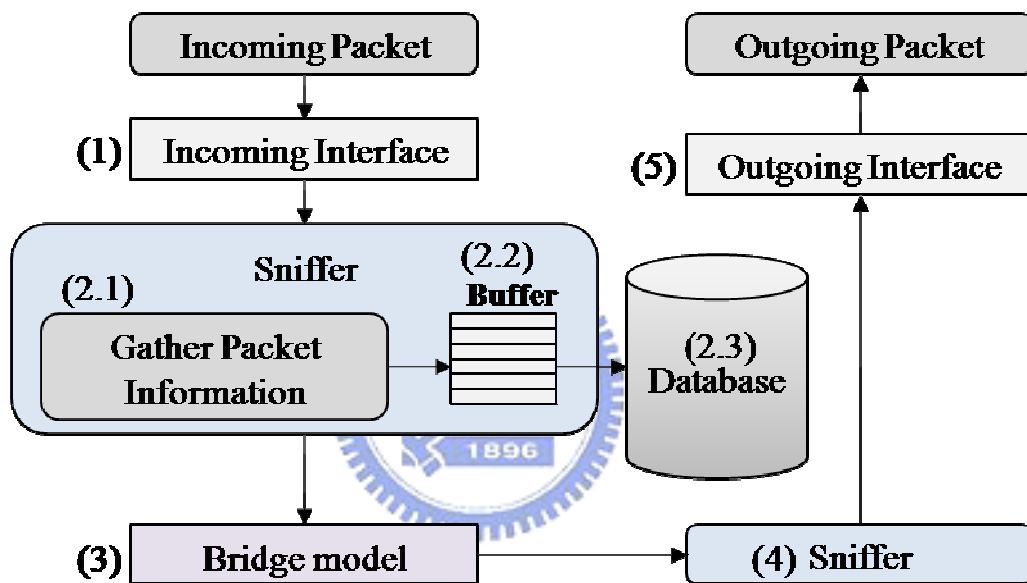


Figure 12 Procedure of packet transmission through sniffer

The process of the sniffer captures packets information is shown in Figure 13. Packet information are recorded after checking Buffer.

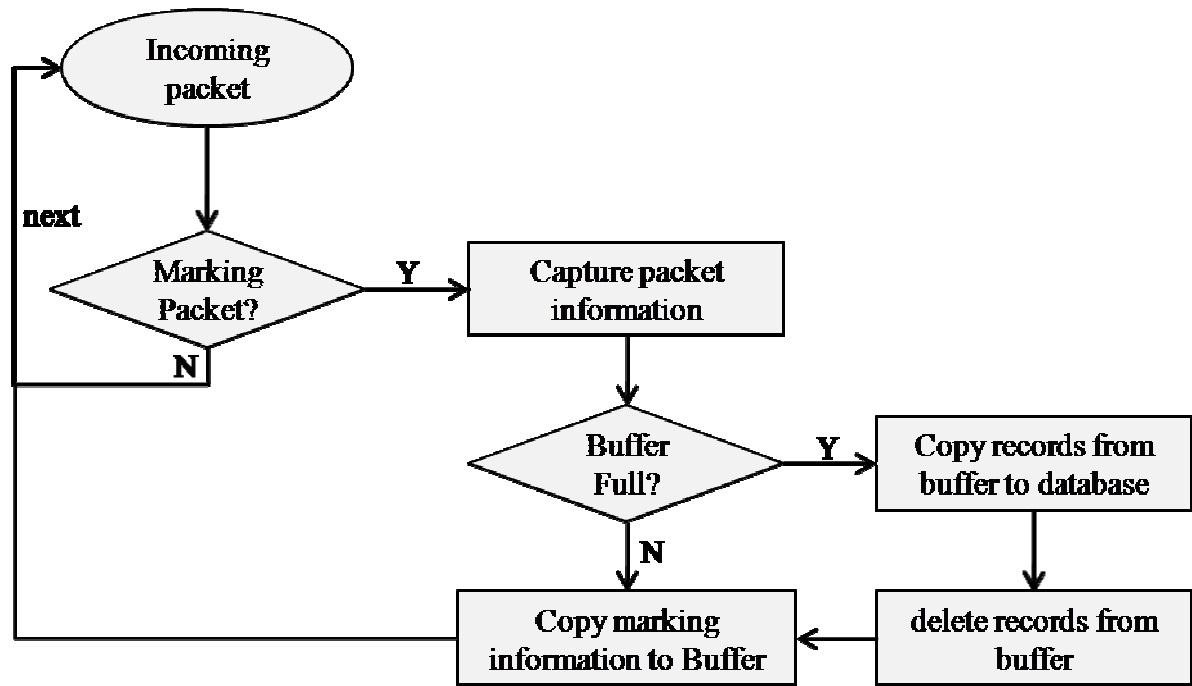


Figure 13 Algorithm of packet logging

3.6.3. The Process of Traceback

The traceback process contains two sides. First, the MCC captured the restriction from the GUI. The conditions is written into the socket and sent to the IPM router. The IPM router received the request message logins the database and searches the records satisfied the conditions. Afterwards, the IPM router gathers the records into data message and returns to the MCC. The MCC analyzes the data message and displays the records in GUI. Figure 14 depicts the protocol between MCC and IPM router.

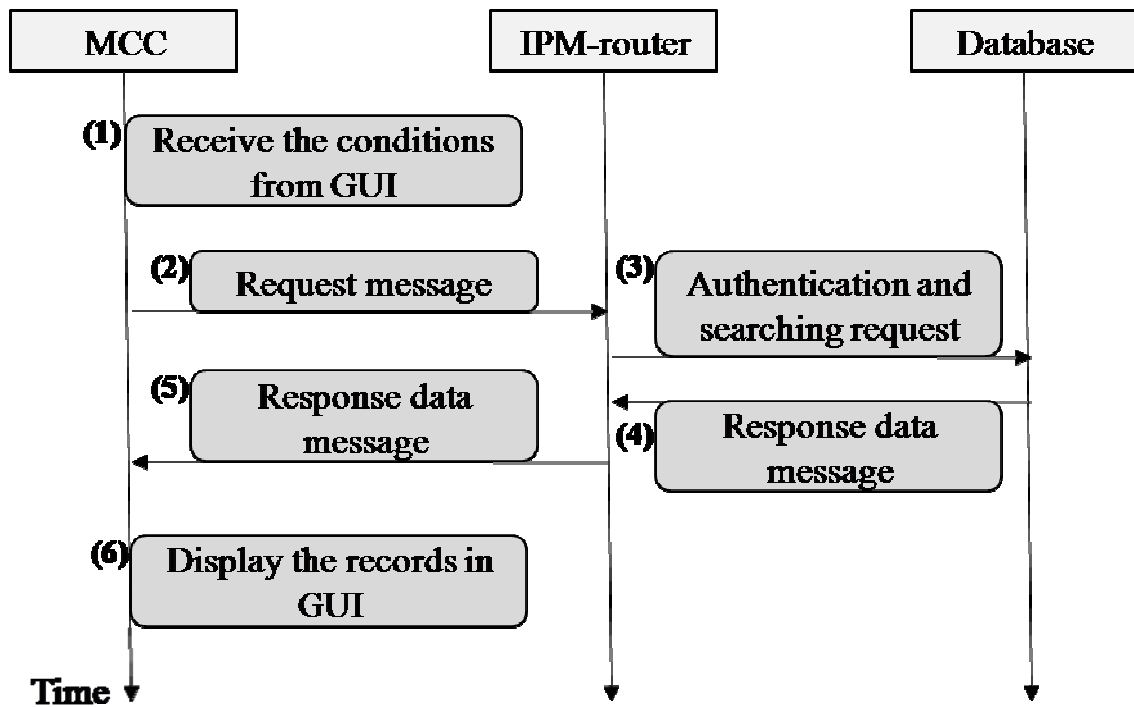


Figure 14 Connection between MCC and IPM router



Chapter 4. Performance Evaluation

4.1. Attack Scenario

The attacker utilizes the IP spoof techniques to transfer the forged packets. The packets are sent to the same destination. The attacker masquerades the source IP address of the neighbor location so that the packets are sent as another host at the victim.

Figure 15 depicts that Attacker from 192.168.64.1 use IP spoofing technique to masquerade SIM (192.168.16.11). Packets from attacker seem to be sent from SIM (192.168.16.11).

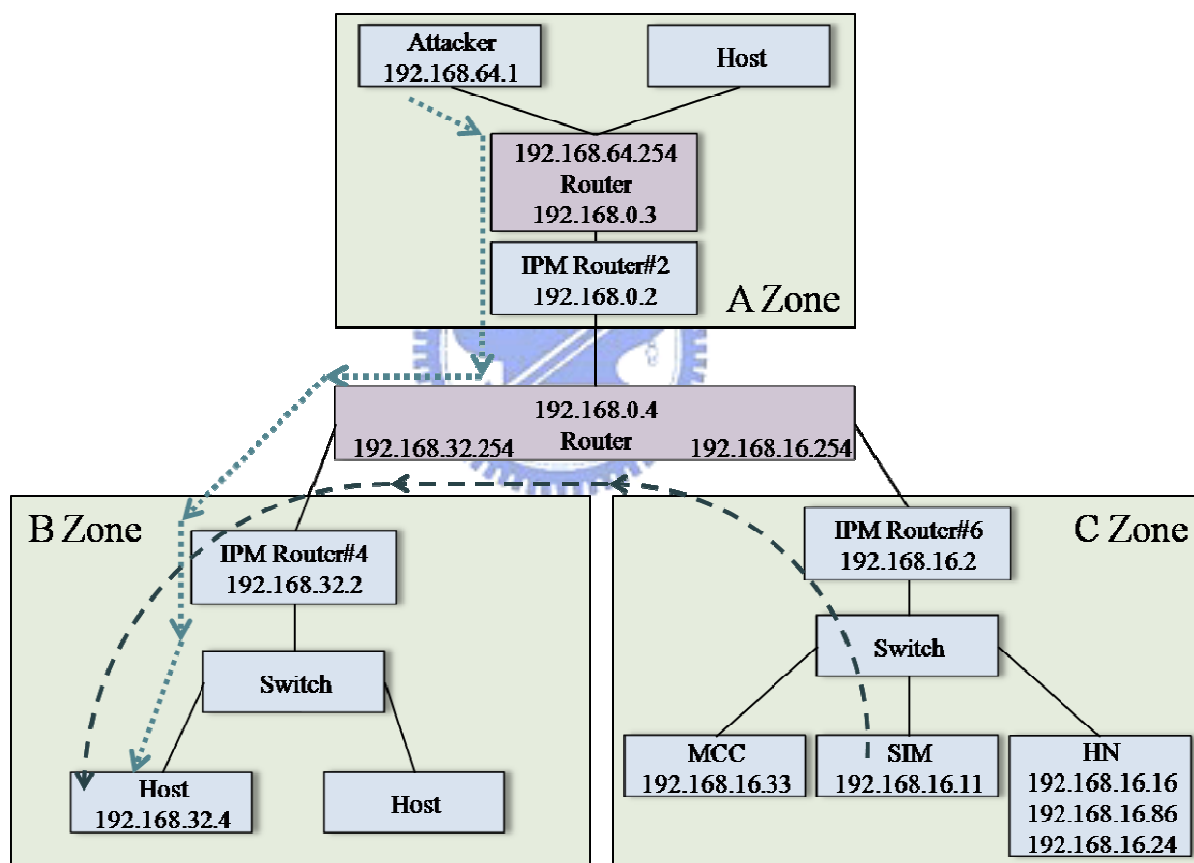


Figure 15 Scenario of packet spoofing from an attack

4.2. Experiment Environment

4.2.1. Hardware

Each IPM router uses the same hardware specification. The details of the equipment are

shown in Table 4. The hardware of MCC is as same as IPM router.

Table 4 Hardware specification

Hardware	Specification
CPU	Intel E7200
Motherboard	Gigabyte EP35-DS3LP35/ICH9
RAM	A-DATA DDRII 800 2GB x 2
VGA	GeForce 7200 series 128M
HDD	WD 6400 AAKS 640GB
PSU	FSP-350W APFC
NIC	Ethernet Cards x 2

4.2.2. Network Topology

IPM routers place in three spaces. The spaces are called A zone, B zone and C zone. Each of the zones has a gateway and a IPM router. The IPM router is adjacent to the gateway and marks the packets. The packets passed through the gateway also go through the IPM router. The network topology is show in Figure 16. There are many PCs connected to normal routers. The three zones are connected to each other.

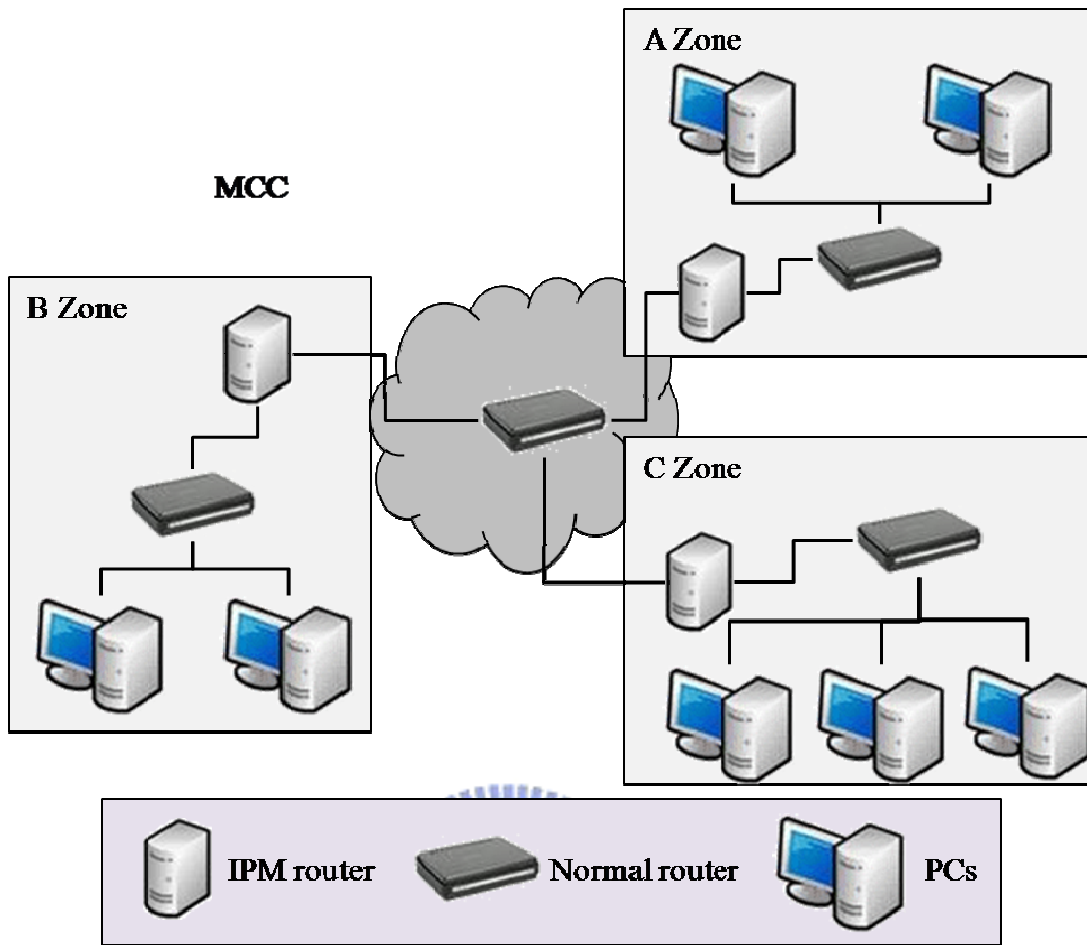


Figure 16 Network topology

4.2.3. Functions

The difference between IPM router and normal router are the process of forwarding, supply of applications and the services. The machines contained functions are shown in Table 5.

The IPM router contains forwarding packets, gathering and analyzing packets, marking packets and searching services. The IPM router receives the packets came from the interface and forwarding to other interface. It also utilizes the sniffer software to gather and analyze these packets. The packets combine IP header with marking information during the forwarding process. The MCC provides GUI searching engine for users to input the conditions. The conditions are sent to IPM router for searching the records gathered from the transmission. Afterwards, the IPM router enables the searching services to search the records from database and sends to the MCC that requires the searching services. The MCC displays

the records to the users. The normal router is working as legacy router.

Table 5 Functions of the machines

Machine	Function
IPM router	Forwarding packets
	Gathering and analyzing packets
	Marking packets
	Searching services
MCC	Providing GUI searching engine
Normal router	Forwarding packets

4.3. Experiment Scenarios

There are three zones in the network. Each zone has two computer, one IPM router and one legacy router. The network setting of each zone is shown in Table 6.

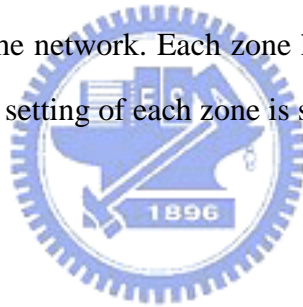


Table 6 Network setting of all equipment

Zone	Equipment	IP	Sub-mask	Gateway
A	Normal router	192.168.64.254	255.255.255.0	X
		192.168.0.3	255.255.255.0	X
	Attacker	192.168.64.1	255.255.255.0	192.168.64.254
	Host	192.168.64.2	255.255.255.0	192.168.64.254
	IPM router [IID:2]	192.168.0.2	255.255.255.0	192.168.0.4
B	Normal router	192.168.32.254	255.255.255.0	X
		192.168.32.254	255.255.255.0	X
	Host	192.168.32.4	255.255.255.0	192.168.32.254
	Host	192.168.32.7	255.255.255.0	192.168.32.254
	IPM router [IID:4]	192.168.32.2	255.255.255.0	192.168.32.254
C	MCC	192.168.16.33	255.255.255.0	X
	SIM	192.168.16.11	255.255.255.0	192.168.16.254
	HN	192.168.16.16	255.255.255.0	192.168.16.254
	IPM router [IID:6]	192.168.16.2	255.255.255.0	192.168.16.254
	Normal router	192.168.0.4	255.255.255.0	X
		192.168.16.254	255.255.255.0	X
		192.168.32.254	255.255.255.0	X

Traffic flows are transmitted through these three zones. IPM routers mark packets passed through and record marking information. According to the records, we could know which zone packets came from and its source IP address. If source IP address belongs to other zone, the result shows that source IP address of the packet is masqueraded.

4.4. Experiment Result

Traceback process is used at the last step. We show the GUI for presenting the results of the wrong and correct. Figure 17 depicts the traceback by traceroute. IP spoofing causes the wrong host and area Figure 18 depicts the correct traceback process by IPM routers.

```

eason@eason-laptop: ~$ traceroute 192.168.16.11
traceroute to 192.168.16.11 (192.168.16.11), 30 hops max, 40 byte packets
 1 192.168.32.254 (192.168.32.254) 0.938 ms 0.281 ms 0.257 ms
 2 192.168.16.11 (192.168.16.11) 0.347 ms 0.358 ms 0.325 ms
eason@eason-laptop: ~$
    
```

Figure 17 Traceroute process by victim

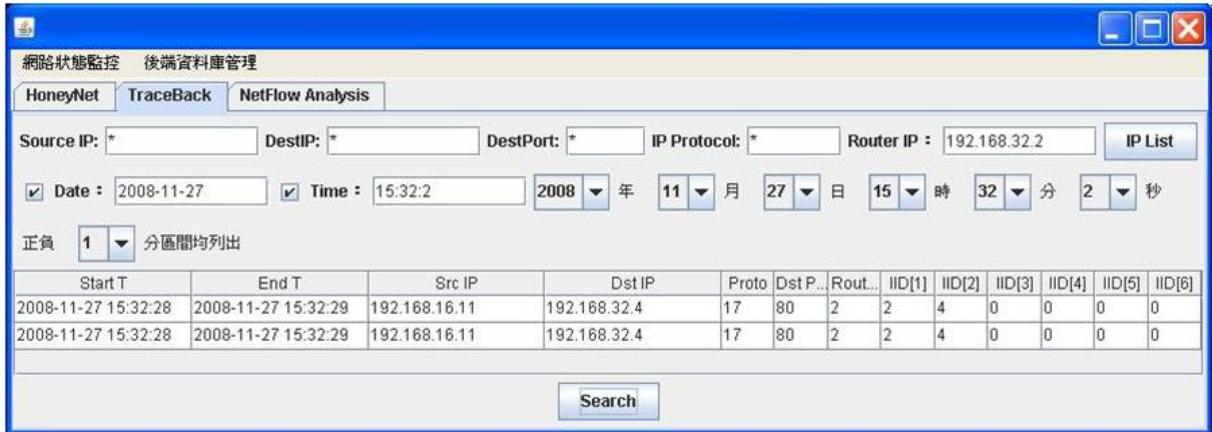


Figure 18 Traceback process by IPM router

The results are sub-divided into five parts. The first part is the setting of the IPM router. The IPM router has to set the initial value for the IID number and the region. The IPM router bases on the setting to decide if the packets need to mark the information. The second part is the execution of the sniffer. The information that the sniffer gathered is shown in the screen. The third part is that the database stores the records and controls on the web. The fourth part is the traceback GUI. The users utilize the GUI to input the conditions and get the records from one of IPM routers. The last part is the real network performance.

4.4.1. Setting Result

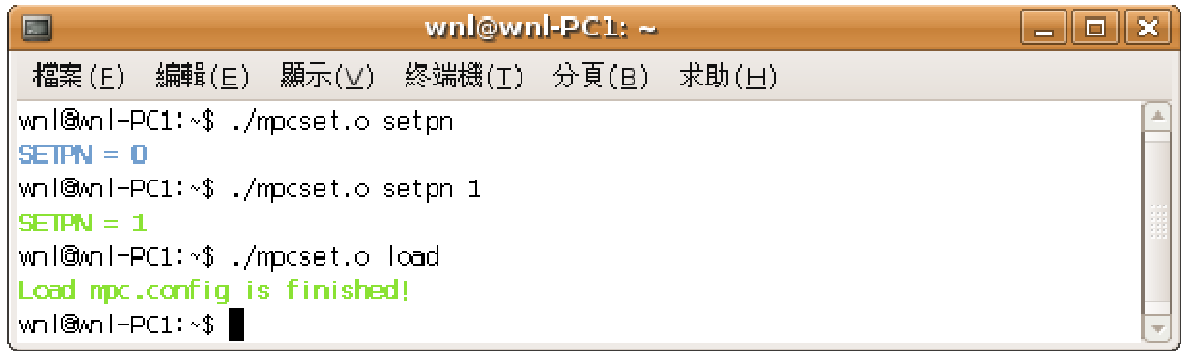
The program we designed could change the variables in the kernel space. We could use this program to identify the IID number, set the domain region and the marking decision. Figure 19 depicts the total commands of the program. The commands are using to change the variables as show in Figure 20. We can use the file to configure the setting. Figure 21 show the execution of the commands.

```
wnl@wnl-PC1: ~  
檔案(E) 編輯(E) 顯示(V) 終端機(T) 分頁(B) 求助(H)  
wnl@wnl-PC1: ~$ ./mpcset.o help  
-----  
Example:  
Show IID Number :      ./mpcset.o iid  
Set IID Number :      ./mpcset.o iid <number 1-511>  
Add ip and submask :   ./mpcset.o rip add <ip>/<submask 0-32>  
Delete ip and submask : ./mpcset.o rip del <ip>/<submask 0-32>  
Show all ip and submask : ./mpcset.o rip show  
Show SETPN Number :   ./mpcset.o setpn  
Set SETPN Number :    ./mpcset.o setpn <number 0-1>  
Load the setting :     ./mpcset.o load  
-----  
wnl@wnl-PC1: ~$ █
```

Figure 19 Commands of the IPM router

```
wnl@wnl-PC1: ~  
檔案(E) 編輯(E) 顯示(V) 終端機(T) 分頁(B) 求助(H)  
wnl@wnl-PC1: ~$ ./mpcset.o iid  
IID = 4  
wnl@wnl-PC1: ~$ ./mpcset.o iid 6  
IID = 6  
wnl@wnl-PC1: ~$ ./mpcset.o rip add 192.168.1.0/24  
Add the record into RIP!  
IP:192.168.1.0 submask:FFFFFF00  
wnl@wnl-PC1: ~$ ./mpcset.o rip show  
The records(Total:1):  
IP:192.168.1.0 submask:FFFFFF00  
wnl@wnl-PC1: ~$ ./mpcset.o rip del 192.168.1.0/24  
Succeed! Delete the record from RIP!  
wnl@wnl-PC1: ~$  
wnl@wnl-PC1: ~$ █
```

Figure 20 Operations of the IPM router



```
wnl@wnl-PC1: ~  
檔案(E) 編輯(E) 顯示(V) 終端機(T) 分頁(B) 求助(H)  
wnl@wnl-PC1: ~$ ./mpcset.o setpn  
SETPN = 0  
wnl@wnl-PC1: ~$ ./mpcset.o setpn 1  
SETPN = 1  
wnl@wnl-PC1: ~$ ./mpcset.o load  
Load npc.config is finished!  
wnl@wnl-PC1: ~$ █
```

Figure 21 Domain control and file loading

4.4.2. Sniffer Result

The sniffer program is setting in each of the interface. The packets are captured by sniffer and analyzed the marking information. The results are shown in Figure 22. Start time and End time are the time on the buffer. The main part is the information of IP Option. The marking information is analyzed by sniffer and separated into different fields.



```
root@wnl-PC1: ~
檔案(E) 編輯(E) 顯示(V) 終端機(T) 分頁(B) 求助(H)
Start: 2009-06-18 02:01:49 End: 2009-06-18 02:01:49
source IP = 0xc0a80102(192.168.1.2) ---> dest IP = 0xe00000fb(224.0.0.251)
Protocol:UDP(17) Source Port:5353 Dest Port:5353
Option=27, Length=8, Hash=237,
Router Number:1 -->IID1=6 ,IID2=0,IID3=0, IID4=0 ,IID5=0,IID6=0

Start: 2009-06-18 02:01:56 End: 2009-06-18 02:01:56
source IP = 0xc0a80102(192.168.1.2) ---> dest IP = 0xc0a801fe(192.168.1.254)
Protocol:TCP(6) Source Port:40129 Dest Port:80
Option=27, Length=8, Hash=207,
Router Number:1 -->IID1=6 ,IID2=0,IID3=0, IID4=0 ,IID5=0,IID6=0

Start: 2009-06-18 02:02:52 End: 2009-06-18 02:02:52
source IP = 0xc0a80102(192.168.1.2) ---> dest IP = 0xe00000fb(224.0.0.251)
Protocol:UDP(17) Source Port:5353 Dest Port:5353
Option=27, Length=8, Hash=237,
Router Number:1 -->IID1=6 ,IID2=0,IID3=0, IID4=0 ,IID5=0,IID6=0

Start: 2009-06-18 02:03:06 End: 2009-06-18 02:03:06
source IP = 0xc0a80102(192.168.1.2) ---> dest IP = 0xc0a801fe(192.168.1.254)
Protocol:TCP(6) Source Port:56429 Dest Port:80
Option=27, Length=8, Hash=35,
Router Number:1 -->IID1=6 ,IID2=0,IID3=0, IID4=0 ,IID5=0,IID6=0
```

Figure 22 Packet information gathered from the sniffer

4.4.3. Database Result

The sniffer captures the marking information and sends the information into the table in the database. The table information is shown by phpMyAdmin. Figure 23 depicts the records in the “tam” table.

ETime	SIP	DIP	Protocol	DPORT	IIDNUM	IID1
2009-06-18 01:35:09	3232235778	3232251905	17	53	1	2
2009-06-18 01:35:08	3232235778	3232251905	6	80	1	2
2009-06-18 01:36:00	3232235778	3232251905	6	80	1	2
2009-06-18 01:35:09	3232235778	3232251905	6	80	1	2
2009-06-18 01:35:09	3232235778	3232251905	6	80	1	2
2009-06-18 01:35:09	3232235778	3232251905	6	80	1	2
2009-06-18 01:35:08	3232251905	3232235778	17	39885	2	6
2009-06-18 01:35:08	3232251905	3232235778	6	53184	2	6
2009-06-18 01:35:08	3232251905	3232235778	17	39494	2	6
2009-06-18 01:35:08	3232251905	3232235778	6	48755	2	6
2009-06-18 01:35:24	3232251905	3232235778	6	48761	2	6
2009-06-18 01:35:24	3232251905	3232235778	6	48762	2	6
2009-06-18 01:35:24	3232251905	3232235778	6	48763	2	6
2009-06-18 01:35:24	3232251905	3232235778	6	48764	2	6

Figure 23 Records in the database

4.4.4. Traceback Result

The traceback result is different from the conditions. The users select the conditions to find the information that they need. Figure 24 depicts that the user choose Source IP and Port to trace back the path so that the result contains the same source IP and port.

Source IP: 192.168.1.2 DestIP: * DestPort: 80 IP Protocol: * Router IP: 192.168.1.3

Date: 2009-6-18 Time: 3:26:16 2009 年 6 月 18 日 3 時 26 分 16 秒

正頁 1 分區間均列出

Start Time	End Time	Source IP	Dest. IP	Protocol	Dest Port	Router Num...	ID[1]	ID[2]	ID[3]	ID[4]	ID[5]
2009-06-18 01:35:08	2009-06-18 01:35:08	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:35:08	2009-06-18 01:36:00	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:09	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:35:09	2009-06-18 01:35:09	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:35:09	2009-06-18 01:35:09	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:36:28	2009-06-18 01:36:29	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:36:29	2009-06-18 01:36:29	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:36:30	2009-06-18 01:36:30	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:37:44	2009-06-18 01:37:44	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:37:44	2009-06-18 01:37:44	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:41:44	2009-06-18 01:41:45	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:41:44	2009-06-18 01:41:45	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:42:52	2009-06-18 01:42:52	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:42:52	2009-06-18 01:43:30	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:45:41	2009-06-18 01:46:00	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0

Search

Figure 24 Traceback with the condition “Source IP and Port”

Figure 25 depicts that the users choose Destination IP and Date to trace back the path. The result show that the same destination with the same date and different destination port.

Source IP: * DestIP: 192.168.1.2 DestPort: * IP Protocol: * Router IP: 192.168.1.3

Date: 2009-6-18 Time: 3:26:15 2009 年 6 月 18 日 3 時 26 分 16 秒

正頁 1 分區間均列出

Start Time	End Time	Source IP	Dest. IP	Protocol	Dest Port	Router Num...	ID[1]	ID[2]	ID[3]	ID[4]	ID[5]
2009-06-18 01:35:08	2009-06-18 01:35:08	192.168.64.1	192.168.1.2	17	39885	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:08	192.168.64.1	192.168.1.2	6	53184	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:08	192.168.64.1	192.168.1.2	17	39494	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:08	192.168.64.1	192.168.1.2	6	48755	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:24	192.168.64.1	192.168.1.2	6	48761	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:24	192.168.64.1	192.168.1.2	6	48762	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:24	192.168.64.1	192.168.1.2	6	48763	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:24	192.168.64.1	192.168.1.2	6	48764	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:08	192.168.64.1	192.168.1.2	17	51774	2	6	2	0	0	0
2009-06-18 01:35:09	2009-06-18 01:35:09	192.168.64.1	192.168.1.2	6	50559	2	6	2	0	0	0
2009-06-18 01:35:09	2009-06-18 01:35:24	192.168.64.1	192.168.1.2	6	48766	2	6	2	0	0	0
2009-06-18 01:35:09	2009-06-18 01:35:09	192.168.64.1	192.168.1.2	17	44859	2	6	2	0	0	0
2009-06-18 01:35:09	2009-06-18 01:35:09	192.168.64.1	192.168.1.2	6	35133	2	6	2	0	0	0
2009-06-18 01:35:09	2009-06-18 01:35:09	192.168.64.1	192.168.1.2	17	36033	2	6	2	0	0	0
2009-06-18 01:35:09	2009-06-18 01:35:09	192.168.64.1	192.168.1.2	6	51742	2	6	2	0	0	0

Search

Figure 25 Traceback with the condition “Destination IP and Date”

Figure 26 depicts that the records with the same date and the time between 3:29:47 to

3:31:47. Time is the region of the value chose by users.

The screenshot shows a network management application window with search filters and a table of results. The filters are set to Date: 2009-6-18 and Time: 1:34:8. The table lists various network events with columns for Start Time, End Time, Source IP, Dest. IP, Protoco, Dest Port, Router Num..., and several ID columns.

Start Time	End Time	Source IP	Dest. IP	Protoco	Dest Port	Router Num...	ID[1]	ID[2]
2009-06-18 01:35:08	2009-06-18 01:35:09	192.168.1.2	192.168.64.1	17	53	1	2	0	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:08	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:35:08	2009-06-18 01:36:00	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:09	192.168.1.2	192.168.64.1	6	80	1	2	0	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:08	192.168.64.1	192.168.1.2	17	39885	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:08	192.168.64.1	192.168.1.2	6	53184	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:08	192.168.64.1	192.168.1.2	17	39494	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:08	192.168.64.1	192.168.1.2	6	48755	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:24	192.168.64.1	192.168.1.2	6	48761	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:24	192.168.64.1	192.168.1.2	6	48762	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:24	192.168.64.1	192.168.1.2	6	48763	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:24	192.168.64.1	192.168.1.2	6	48764	2	6	2	0	0	0
2009-06-18 01:35:08	2009-06-18 01:35:08	192.168.64.1	192.168.1.2	17	51774	2	6	2	0	0	0

Figure 26 Traceback with the condition “Date and Time”



Chapter 5. Conclusion and Future Work

Developing a traceback system that can trace a single packet has been viewed as impractical due to the tremendous storage requirements of saving packet data. We believe that the implementation of IPM router is feasible for tracing a single packet. Our system is based on the observation that the marking information under attack would discover the attack path.

Our system contains three schemes for implementation. In the marking scheme, we utilize the identifiable number to reduce the space of the option. Additionally, we use MD5 function to hash a number for verification of the fields. Attackers have to try the correct hash number for masquerading option fields. The marking scheme marks packets according the domain value of RIP setting. We could choose networks that we want to mark or not. In the logging scheme, we use buffer space to reduce same records and store them into local database. Same packet information gathers into one record during a moment. In the traceback scheme, we could find the area that packets belong to according the records. The records show the IID information so that we could transfer IID to normal IP address to know the area. Packets with wrong address are discovered by comparing the area and IP address.

An advantage of our system is that it works in real-time and non-real-time and traces a single packet. No matter how attackers modify the source IP address, the area that packets come from can not be hidden.

Commercial firewalls filter out packets by rules set by management. Packets with marking information may drop by firewall so that the transmission is not complete and failure. In the future, the marking information may put into other header or fields which are infrequent used. The database of each IPM router could interact for changing marking information so that the whole routing path would discover.

IPM would combine with other technique for traceback in wireless network. Access points (AP) in wireless network should keep the connection information during connecting to them such that the IPM could traceback to the AP and AP applies MAC address to know who

uses this IP address. APs are the roles of monitoring all information of mobile stations.



References

- [1] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Network Support for IP Traceback," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 226-237, 2001.
- [2] S. Deering, "Internet Protocol, Version 6 IPv6," *RFC 2460*, 1998.
- [3] A. Belenky and N. Ansari, "IP Traceback With Deterministic Packet Marking," *IEEE Communication Letters*, vol. 7, pp. 162-164, Apr. 2003.
- [4] A. Belenky and N. Ansari, "Tracing multiple attackers with deterministic packet marking (DPM)," in *Proceedings of IEEE Pacific Rim Con. Communications, Computers and Signal Processing*, vol. 1, pp. 49-52, Aug. 2003.
- [5] R. Chen, J. Park, and R. Marchany, "RIM: Router Interface Marking for IP Traceback," in *Proceedings of IEEE GLOBECOM*, pp. 1-5, Nov. 2006.
- [6] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, B. Schwartz, S. Kent, and W. Strayer, "Single-Packet IP Traceback," *IEEE/ACM Transactions on Networking*, vol. 10, no. 6, pp. 721-734, 2002.
- [7] D. Basheer and G. Manimaran, "Novel hybrid schemes employing packet marking and logging for IP traceback," *IEEE Trans. Parallel and Distributed Systems*, Vol. 17(5), pp. 403– 418, May 2006.
- [8] S. Bellovin, M. Leech, and T. Taylor, ICMP Traceback Messages, Internet Draft, draft-ietf-itrace-04.txt, Feb. 2003.
- [9] A. Yaar, A.Perrig, and D.Song, "FIT: Fast Internet Traceback," in *Proceedings of INFOCOM*, Mar. 2005, pp. 1395–1406.

Appendix A. Codes

Code 1 mpcset.c	39
Code 2 mylistener.c	46
Code 3 myd.c	58
Code 4 Traceback.java	60
Code 5 br_forward.c	68

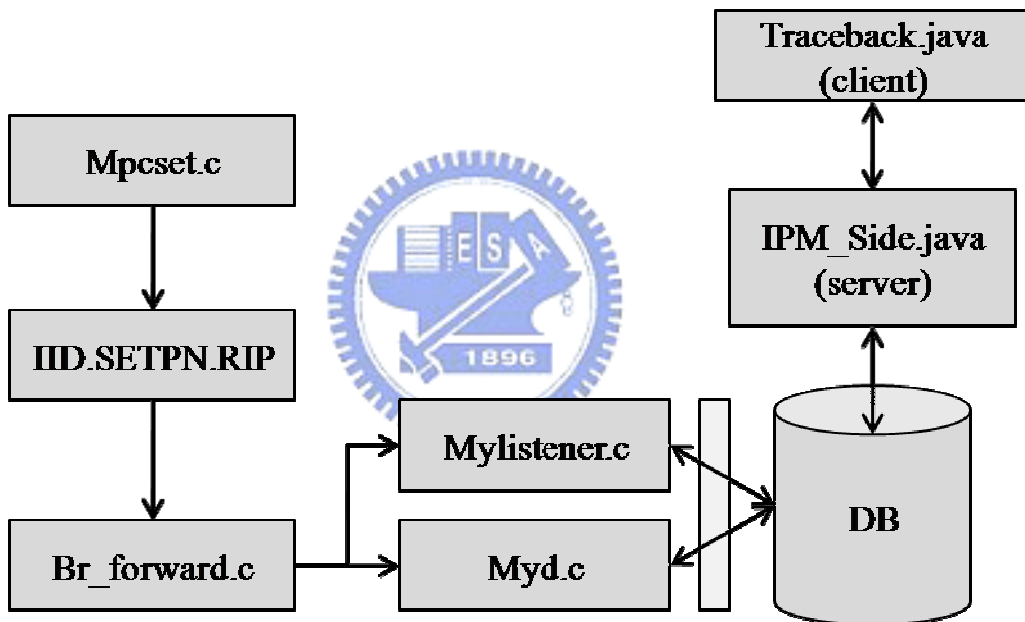


Figure 27 Association

Code 1 `mpcset.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <syscall.h>

#define RED      "\E[31m\E[1m"
#define GREEN   "\E[32m\E[1m"
#define BLUE    "\E[34m\E[1m"
#define NORMAL  "\E[m"

unsigned int reverse_submask(unsigned int num){
    int i;
    unsigned int submask;
    submask=0;
    for(i=31;i>=0;i--){
        if(num%2==1)
            submask += 1<<i;
        num = num>>1;
    }
    return submask;
}

void savefile(){
    FILE *output;
    int i;
    int temp;

    if((output=fopen("mpc.config","w+"))==NULL){
        printf("File mpc.config is not writeable!\n");
        return;
    }
    // save IID first
    // save SETPN second
    // save RIP records
    fprintf(output,"%d\n",syscall(__NR_getIID));
    fprintf(output,"%d\n",syscall(__NR_getSETPN));
    temp = syscall(__NR_getCNT);
    for(i=1;i<=temp;i++){
        fprintf(output,"%d/%d\n",syscall(__NR_getRIP,i),syscall(__NR_getSUB,i));
    }
    fclose(output);
}

void loadfile(){
    FILE *input;
    int cnt;
    int iid;
    int setpn;
    unsigned int ip;
```




```

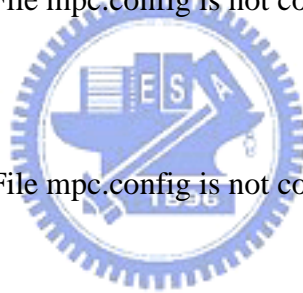
unsigned int submask;

if((input=fopen("/home/wnl/mpc.config","r"))==NULL){
    fprintf(stderr,RED"File mpc.config is not found!\n"NORMAL);
    exit(-1);
}

// read IID first
if(fscanf(input,"%d\n",&iid)==EOF){
    fprintf(stderr,RED"File mpc.config is not correct context!\n"NORMAL);
    fclose(input);
    exit(-1);
}
if(iid<1 || iid>511){
    fprintf(stderr,RED"File mpc.config is not correct context!\n"NORMAL);
    fclose(input);
    exit(-1);
}
syscall(__NR_setIID,iid);
// read SETPN second
if(fscanf(input,"%d\n",&setpn)==EOF){
    fprintf(stderr,RED"File mpc.config is not correct context!\n"NORMAL);
    fclose(input);
    exit(-1);
}
if(setpn<0 || setpn>1){
    fprintf(stderr,RED"File mpc.config is not correct context!\n"NORMAL);
    fclose(input);
    exit(-1);
}
syscall(__NR_setSETPN,setpn);
// clean RIP all records
syscall(__NR_setCNT,0);
cnt=0;
// read RIP records
while(fscanf(input,"%d/%d\n",&ip,&submask)!=EOF){
    cnt++;
    syscall(__NR_setRIP,ip,cnt);
    syscall(__NR_setSUB,submask,cnt);
    syscall(__NR_setCNT,cnt);
}
fclose(input);
printf(GREEN"Load mpc.config is finished!\n"NORMAL);
}

void IID(unsigned int iid){
    if(iid < 1 || iid > 511){
        fprintf(stderr,RED"IID Number out of range (1-511)\n"NORMAL);
        exit(-1);
    }
}

```



```

syscall(__NR_setIID,iid);
printf(GREEN"IID = %d\n"NORMAL,syscall(__NR_getIID));
savefile();
}

void RIP(int modes,int argc,char **argv){
char *ip_str;
char *submask_str;
unsigned int ip;
unsigned int temp;
int count;
int num;
unsigned int submask;
unsigned int submask_2;

// param[0] = add, del or show
if(modes==1){
// add
if(argc<2){
printf(stderr,RED"Too few parameter: rip add
<ip/submask>\n"NORMAL);
exit(-1);
}
// divide ip and submask
ip_str=strtok(argv[1],"/");
submask_str=strtok(NULL,"/");

// deal with IP
ip_str=strtok(ip_str,".");
ip = 0;
count = -8;
while(ip_str != NULL){
count += 8;
temp = atoi(ip_str);
if(temp > 255 || temp < 0){
printf(stderr,RED"Error : IP address is not correct!\n"NORMAL);
exit(-1);
}
ip += (temp << count);
ip_str = strtok(NULL,".");
}
if(count != 24){
printf(stderr,RED"Error : IP address is not correct!\n"NORMAL);
exit(-1);
}
// deal with submask
submask_2 = atoi(submask_str);
submask=0;
while(submask_2>0){
submask = submask*2 + 1;

```



```

        submask_2--;
    }
    temp = syscall(__NR_getCNT);
    if(temp>=30){
        fprintf(stderr,RED"Error : The records are full! Please delete record
first!\n"NORMAL);
        exit(-1);
    }
    temp++;
    syscall(__NR_setRIP,ip,temp);
    syscall(__NR_setSUB,submask,temp);
    syscall(__NR_setCNT,temp);
    savefile();
    printf(GREEN"Add the record into RIP!\n");

printf("IP:%d.%d.%d.%d\t",ip&0xFF,ip>>8&0xFF,ip>>16&0xFF,ip>>24&0xFF);
    printf("submask:%08X\n"NORMAL,reverse_submask(submask));
}else if(modes==2){
    // del
    if(argc<2){
        fprintf(stderr,RED"Too few parameter: rip del <ip/submask>\n"NORMAL);
        exit(-1);
    }
    // divide ip and submask
    ip_str=strtok(argv[1],"/");
    submask_str=strtok(NULL,"/");

    // deal with IP
    ip_str=strtok(ip_str,".");
    ip = 0;
    count = -8;
    while(ip_str != NULL){
        count += 8;
        temp = atoi(ip_str);
        if(temp > 255 || temp < 0){
            fprintf(stderr,RED"Error : IP address is not correct!\n"NORMAL);
            exit(-1);
        }
        ip += (temp << count);
        ip_str = strtok(NULL,".");
    }
    if(count != 24){
        fprintf(stderr,RED"Error : IP address is not correct!\n"NORMAL);
        exit(-1);
    }
    // deal with submask
    submask_2 = atoi(submask_str);
    submask=0;
    while(submask_2>0){
        submask = submask*2 + 1;

```



```

        submask_2--;
    }
    temp = syscall(__NR_getCNT);

    //search the records
    num = 1;
    while(temp >= num){
        if(syscall(__NR_getRIP,num)==ip &&
syscall(__NR_getSUB,num)==submask){
            break;
        }
        num++;
    }
    if(num>temp){
        fprintf(stderr,RED"Error : The record is not found!\n"NORMAL);
        exit(-1);
    }
    ip=syscall(__NR_getRIP,temp);
    submask=syscall(__NR_getSUB,temp);
    syscall(__NR_setRIP,ip,num);
    syscall(__NR_setSUB,submask,num);
    temp--;
    syscall(__NR_setCNT,temp);
    savefile();
    printf(GREEN"Succeed! Delete the record from RIP!\n"NORMAL);
}else if(modes==3){
    // show
    temp = syscall(__NR_getCNT);
    printf(GREEN"The records(Total:%d):\n",temp);
    for(num=1; num<=temp; num++){
        ip=syscall(__NR_getRIP,num);
        submask=syscall(__NR_getSUB,num);

        printf("IP:%d.%d.%d.%d\t",ip&0xFF,ip>>8&0xFF,ip>>16&0xFF,ip>>24&0xFF);
        printf("submask:%08X\n",reverse_submask(submask));
    }
    printf(NORMAL);
}
}

void SETPN(unsigned int setpn){
    if(setpn < 0 || setpn > 1){
        fprintf(stderr,RED"SETPN Number out of range (0-1)\n"NORMAL);
        exit(-1);
    }
    syscall(__NR_setSETPN,setpn);
    savefile();
    printf(GREEN"SETPN = %d\n"NORMAL,setpn);
}

```

```

int main(int argc, char **argv)
{
    char *cmds[]={ "iid", "rip", "setpn", "load", "help" };
    char *ripcmds[]={ "add", "del", "show" };
    int modes, ripmodes;

    if(argc < 2)
    {
        fprintf(stderr, RED "%s <execute command> <parameter>\n" NORMAL, argv[0]);
        return -1;
    }

    if(!strcasecmp(cmds[0], argv[1])){
        // iid
        modes=1;
    }else if(!strcasecmp(cmds[1], argv[1])){
        // rip
        modes=2;
    }else if(!strcasecmp(cmds[2], argv[1])){
        // setpn
        modes=3;
    }else if(!strcasecmp(cmds[3], argv[1])){
        // load config
        loadfile();
        return 0;
    }else if(!strcasecmp(cmds[4], argv[1])){
        // help
        printf("-----\n");
        printf("Example:\n");
        printf("\tShow IID Number :          %s iid\n", argv[0]);
        printf("\tSet IID Number :          %s iid <number 1-511>\n", argv[0]);
        printf("\tAdd ip and submask :        %s rip add <ip>/<submask
0-32>\n", argv[0]);
        printf("\tDelete ip and submask :     %s rip del <ip>/<submask
0-32>\n", argv[0]);
        printf("\tShow all ip and submask : %s rip show\n", argv[0]);
        printf("\tShow SETPN Number :       %s setpn\n", argv[0]);
        printf("\tSet SETPN Number :        %s setpn <number 0-1>\n", argv[0]);
        printf("\tLoad the setting :         %s load\n", argv[0]);
        printf("-----\n");
        return 0;
    }else{
        fprintf(stderr, RED "%s <execute command> <parameter>\n" NORMAL, argv[0]);
        fprintf(stderr, RED "%s %s:Unknow\n" NORMAL, argv[0], argv[1]);
        return -1;
    }

    switch(modes){
        case 1:
            if(argc < 3){

```

```

        // show IID
        printf(BLUE"IID = %d \n"NORMAL,syscall(__NR_getIID));
        return 0;
    }else{
        IID(atoi(argv[2]));
    }
    break;
case 2:
    if(argc <3){
        fprintf(stderr,RED"%s rip [\"add <ip>/<submask 0-32>\" | \"del
<ip>/<submask 0-32>\" | \"show\"]\n"NORMAL, argv[0]);
        return -1;
    }
    if(!strcasecmp(ripcmds[0],argv[2])){
        // add
        ripmodes=1;
    }else if(!strcasecmp(ripcmds[1],argv[2])){
        // del
        ripmodes=2;
    }else if(!strcasecmp(ripcmds[2],argv[2])){
        // show
        ripmodes=3;
    }else{
        fprintf(stderr,RED"%s rip [\"add <ip>/<submask 0-32>\" | \"del
<ip>/<submask 0-32>\" | \"show\"]\n"NORMAL, argv[0]);
        fprintf(stderr,RED"%s rip %s:Unknow\n"NORMAL, argv[0],argv[2]);
        return -1;
    }
    RIP(ripmodes,argc-2,&argv[2]);
    break;
case 3:
    if(argc <3){
        // show SETPN
        printf(BLUE"SETPN = %d \n"NORMAL,syscall(__NR_getSETPN));
        return 0;
    }else{
        SETPN(atoi(argv[2]));
    }
    break;
default:
    break;
}
return 0;
}

```

Code 2 **mylistener.c**

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <fcntl.h>
#include <netpacket/packet.h>
#include <net/if.h>
#include <net/if_arp.h>
#include <netinet/in.h>
#include <net/ethernet.h>
#include <netinet/ether.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <netinet/tcp.h>
#include <linux/if_ether.h>
#include <arpa/inet.h>
#include <sys/ioctl.h>
#include <unistd.h>
#include <time.h>
#include <sys/time.h>
#include <signal.h>
#include <mysql/mysql.h>
#include <linux/unistd.h>

#define RED          "\E[31m\E[1m"
#define GREEN       "\E[32m\E[1m"
#define YELLOW      "\E[33m\E[1m"
#define BLUE        "\E[34m\E[1m"
#define NORMAL      "\E[m"
#define MAX_BUFFER 100 // max number of records
#define MAX_SECOND 60.0 // Time of life for each record

// The option from internet without editing, only get the information and copy to this
// structure
typedef struct Tempop {
    unsigned short option:8, length:8;
    unsigned char ops[6];
} Tempop;

// The option corss Tempop structure and get the correct information to each field
typedef struct Myop {
    unsigned short option:8, length:8;
    unsigned short hash;
    unsigned short IID[6];
} Myop;
```



```

// Full information for each packet
typedef struct ops {
    time_t Ts,Te; // time of first packet crossed and time of last packet crossed (Same
info.)
    unsigned int source_IP; // Source IP
    unsigned int dest_IP; // Destination IP
    unsigned short protocol; // IP Protocol
    unsigned int source_PORT; // Source Port
    unsigned int dest_PORT; // Destination Port
    unsigned short IID_Num; // Number of IIDs
    Myop op; // Packet Option Information
    struct ops *next,*pre; // Linking list according to time (H:earlist T:latest)
    struct ops *IID_next,*IID_pre; // Linking list according to number of IIDs
}OPs;

// Global variables
int RecordNum; // count number of information
OPs *IIDListH[6], *IIDListT[6]; // IID linking list Head and Tail
OPs *TimeListH,*TimeListT; // Time linking list Head and Tail
MYSQL mysql;
char *host;
char *database;
char *user;
char *passwd;
unsigned int IID;

int Get_IfaceIndex(int fd, const char* interfaceName)
{
    struct ifreq ifr;
    if (interfaceName == NULL)
    {
        return -1;
    }
    memset(&ifr, 0, sizeof(ifr));
    strcpy(ifr.ifr_name, interfaceName);
    if (ioctl(fd, SIOCGIFINDEX, &ifr) == -1)
    {
        printf("RED ioctl error\n");
        return -1;
    }
    return ifr.ifr_ifindex;
}

int set_Iface_promisc(int fd, int dev_id)
{
    struct packet_mreq mr;
    memset(&mr,0,sizeof(mr));
    mr.mr_ifindex = dev_id;
    mr.mr_type = PACKET_MR_PROMISC;
}

```




```

if(setsockopt(fd, SOL_PACKET,
PACKET_ADD_MEMBERSHIP,&mr,sizeof(mr))== -1)
{
    fprintf(stderr,"GREEN set promisc failed! \n");
    return -1;
}
return 0;
}

int compareBuf(OPs *CP1, OPs *CP2){
    if(CP1->source_IP == CP2->source_IP && CP1->dest_IP == CP2->dest_IP)
        if(CP1->dest_PORT == CP2->dest_PORT && CP1->protocol ==
CP2->protocol)
            if(CP1->op.IID[0] == CP2->op.IID[0] &&CP1->op.IID[1] ==
CP2->op.IID[1] &&CP1->op.IID[2] == CP2->op.IID[2] &&CP1->op.IID[3] ==
CP2->op.IID[3] )
                return 1;
    return 0;
}

void usage(char *exename)
{
    fprintf(stderr,RED"%s <interface>\n"NORMAL, exename);
}

void printPacket(OPs *Opbuf)
{
    struct tm sts,ste;

#ifdef SunOS
    memcpy(&sts, localtime(&Opbuf->Ts), sizeof(struct tm));
    memcpy(&ste, localtime(&Opbuf->Te), sizeof(struct tm));
#else
    localtime_r(&Opbuf->Ts, &sts);
    localtime_r(&Opbuf->Te, &ste);
#endif

    fprintf(stdout,"Start: %04d-%02d-%02d %02d:%02d:%02d
",sts.tm_year+1900,sts.tm_mon+1,sts.tm_mday,sts.tm_hour,sts.tm_min,sts.tm_sec);
    fprintf(stdout,"End: %04d-%02d-%02d
%02d:%02d:%02d\n",ste.tm_year+1900,ste.tm_mon+1,ste.tm_mday,ste.tm_hour,ste.tm_min,ste.tm_sec);
    fprintf(stdout,"source IP = 0x%08x",Opbuf->source_IP);
    fprintf(stdout,"(%d.%d.%d.%d)",Opbuf->source_IP>>24&0xFF,Opbuf->source_IP>>
16&0xFF,Opbuf->source_IP>>8&0xFF,Opbuf->source_IP&0xFF);
    fprintf(stdout," ---> ");
    fprintf(stdout,"dest IP = 0x%08x",Opbuf->dest_IP);
    fprintf(stdout,"(%d.%d.%d.%d)\n",Opbuf->dest_IP>>24&0xFF,Opbuf->dest_IP>>16
&0xFF,Opbuf->dest_IP>>8&0xFF,Opbuf->dest_IP&0xFF);
    if(Opbuf->protocol == 6)

```



```

        fprintf(stdout,"Protocol:TCP(%d) ",Opbuf->protocol);
        if(Opbuf->protocol == 17)
            fprintf(stdout,"Protocol:UDP(%d) ",Opbuf->protocol);
            fprintf(stdout,"Source Port:%d Dest Port:%d\n",Opbuf->source_PORT,
Opbuf->dest_PORT);
            fprintf(stdout,"Option=%d, Length=%d, Hash=%d,\n",Opbuf->op.option,
Opbuf->op.length, Opbuf->op.hash);
            fprintf(stdout," Router Number:%d -->",Opbuf->IID_Num);
            fprintf(stdout,"IID1=%d ,IID2=%d,IID3=%d,
",Opbuf->op.IID[0],Opbuf->op.IID[1],Opbuf->op.IID[2]);
            fprintf(stdout,"IID4=%d ,IID5=%d,IID6=%d\n\n",Opbuf->op.IID[3],Opbuf->op.IID[4
],Opbuf->op.IID[5]);
        }

void PacketRecv()
{
    int i;
    time_t nt,nowt;
    struct tm sts,ste;
    OPs *tbuf;
    char *query;

    // get the time and date
    time(&nt);
    memcpy(&nowt,&nt,sizeof(time_t));
    tbuf = TimeListH;

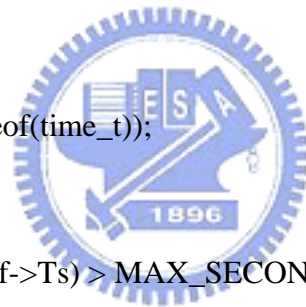
    while(tbuf != NULL){
        if(difftime(nowt,tbuf->Ts) > MAX_SECOND)
        {
            #ifdef SunOS
                memcpy(&sts, localtime(&tbuf->Ts), sizeof(struct tm));
                memcpy(&ste, localtime(&tbuf->Te), sizeof(struct tm));
            #else
                localtime_r(&tbuf->Ts, &sts);
                localtime_r(&tbuf->Te, &ste);
            #endif

            query = malloc(256*sizeof(char));
            sprintf(query,"insert into
tam(Stime,ETIME,SIP,DIP,Protocol,DPORT,IIDNUM,IID1,IID2,IID3,IID4,IID5,IID6) \
value('%04d-%02d-%02d %02d:%02d:%02d','%04d-%02d-%02d
%02d:%02d:%02d',0x%08x,0x%08x,%d,%d,%d,%d,%d, \

%d,%d,%d,%d)",sts.tm_year+1900,sts.tm_mon+1,sts.tm_mday,sts.tm_hour,sts.tm_mi
n,sts.tm_sec, \

ste.tm_year+1900,ste.tm_mon+1,ste.tm_mday,ste.tm_hour,ste.tm_min,ste.tm_sec,tbuf
->source_IP,tbuf->dest_IP,tbuf->protocol, \

```



```

    tbuf->dest_PORT,tbuf->IID_Num,tbuf->op.IID[0],tbuf->op.IID[1],tbuf->op.IID[2],tb
uf->op.IID[3],tbuf->op.IID[4],tbuf->op.IID[5]);
        if(mysql_real_query(&mysql,query,strlen(query))) {

            if(!mysql_real_connect(&mysql,host,user,passwd,database,0,NULL,0)){
                fprintf(stderr, "Failed to connect to database: Error: %s\n",
mysql_error(&mysql));
                return ;
            }

        }

        // delete the record
        TimeListH = TimeListH->next;
        if(TimeListH == NULL)
            TimeListT = NULL;
        else
            TimeListH->pre = NULL;

        if(tbuf->IID_pre == NULL){
            // it's head
            IIDListH[tbuf->IID_Num-1] =
IIDListH[tbuf->IID_Num-1]->IID_next;
            if(IIDListH[tbuf->IID_Num-1] == NULL)
                // no data
                IIDListT[tbuf->IID_Num-1] = NULL;
            else
                IIDListH[tbuf->IID_Num-1] -> IID_pre = NULL;
        }else{
            if(tbuf->IID_next == NULL){
                // it's tail
                IIDListT[tbuf->IID_Num-1] = tbuf->IID_pre;
                IIDListT[tbuf->IID_Num-1]->IID_next = NULL;
            }else{
                // it's middle
                tbuf->IID_pre->IID_next = tbuf->IID_next;
                tbuf->IID_next->IID_pre = tbuf->IID_pre;
            }
        }
        RecordNum--;
        free(tbuf);
    }else
        break;

    tbuf = tbuf->next;
}
}

void recordMAX()
{

```

```

OPs *tbuf; // options pointer
struct tm sts,ste; // structure of time
int i;
char *query;

tbuf = TimeListH;

#ifdef SunOS
    memcpy(&sts, localtime(&tbuf->Ts), sizeof(struct tm));
    memcpy(&ste, localtime(&tbuf->Te), sizeof(struct tm));
#else
    localtime_r(&tbuf->Ts, &sts);
    localtime_r(&tbuf->Te, &ste);
#endif

query = malloc(256*sizeof(char));
sprintf(query,"insert into
tam(Stime,ETIME,SIP,DIP,Protocol,DPORT,IIDNUM,IID1,IID2,IID3,IID4,IID5,IID6) \
value('%04d-%02d-%02d %02d:%02d:%02d','%04d-%02d-%02d
%02d:%02d:%02d',0x%08x,0x%08x,%d,%d,%d,%d,%d, \
%d,%d,%d,%d)",sts.tm_year+1900,sts.tm_mon+1,sts.tm_mday,sts.tm_hour,sts.tm_mi
n,sts.tm_sec, \
ste.tm_year+1900,ste.tm_mon+1,ste.tm_mday,ste.tm_hour,ste.tm_min,ste.tm_sec,tbuf
->source_IP,tbuf->dest_IP,tbuf->protocol, \
tbuf->dest_PORT,tbuf->IID_Num,tbuf->op.IID[0],tbuf->op.IID[1],tbuf->op.IID[2],tb
uf->op.IID[3],tbuf->op.IID[4],tbuf->op.IID[5]);
if(mysql_real_query(&mysql,query,strlen(query))){
    if(!mysql_real_connect(&mysql,host,user,passwd,database,0,NULL,0)){
        fprintf(stderr, "Failed to connect to database: Error: %s\n",
mysql_error(&mysql));
        return;
    }
}

// delete the record
TimeListH = TimeListH->next;
if(TimeListH == NULL)
    TimeListT = NULL;
else
    TimeListH->pre = NULL;

if(tbuf->IID_pre == NULL){
    // it's head
    IIDListH[tbuf->IID_Num-1] = IIDListH[tbuf->IID_Num-1]->IID_next;
    if(IIDListH[tbuf->IID_Num-1] == NULL)
        // no data
        IIDListT[tbuf->IID_Num-1] = NULL;
    else
        IIDListH[tbuf->IID_Num-1] -> IID_pre = NULL;
}

```

```

}else{
    if(tbuf->IID_next == NULL){
        // it's tail
        IIDListT[tbuf->IID_Num-1] = tbuf->IID_pre;
        IIDListT[tbuf->IID_Num-1]->IID_next = NULL;
    }else{
        // it's middle
        tbuf->IID_pre->IID_next = tbuf->IID_next;
        tbuf->IID_next->IID_pre = tbuf->IID_pre;
    }
}
RecordNum--;
free(tbuf);
}

int main(int argc, char **argv)
{
    int listen_fd;
    int ipak=0,maxk=0;
    char buffer[256];
    int frmlen;
    int i;
    sigset_t intmask,oldmask;
    Ttempop *top; // option pointer to packet
    OPs *Opbuf; // packet buffer for record
    OPs *tbuf; // pointer used for linking list
    time_t t; // time
    struct sockaddr_ll sll;
    struct ether_header *eptr; /* net/ethernet.h */
    struct iphdr *ip; // for ip header
    struct tcphdr *tcp; // for tcp header
    struct udphdr *udp; // for udp header
    struct tm sts,ste; // structure of time
    struct itimerval value;

    u_short ether_type;

    if(argc <2)
    {
        usage(argv[0]);
        return -1;
    }

    listen_fd = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
    sll.sll_family = AF_PACKET;
    sll.sll_ifindex = Get_IfaceIndex(listen_fd,argv[1]);
    sll.sll_protocol = htons(ETH_P_ALL);

    if(bind(listen_fd,(struct sockaddr *)&sll,sizeof(sll))== -1)

```

```

{
    fprintf(stderr, YELLOW "bind error:%s !\n" NORMAL, strerror(errno));
    goto FAIL;
}

if(set_iface_promisc(listen_fd, sll.sll_ifindex) == -1)
{
    fprintf(stderr, "BLUE set promisc failed !\n");
    goto FAIL;
}

// read IID
IID = syscall(__NR_getIID);

if(argc>2)
    maxk = atoi(argv[2]);

RecordNum = 0;
signal(SIGALRM, PacketRecv);
value.it_value.tv_sec = 5;
value.it_value.tv_usec = 0;
value.it_interval.tv_sec = 5;
value.it_interval.tv_usec = 0;
setitimer(ITIMER_REAL, &value, NULL);

for(i=0; i<6; i++)
{
    IIDListH[i] = NULL;
    IIDListT[i] = NULL;
}
TimeListH = NULL;
TimeListT = NULL;

sigemptyset(&intmask);
sigaddset(&intmask, SIGALRM);

host = "127.0.0.1";
user="wnl";
passwd="1234";
database="wnl";
mysql_init(&mysql);
if(!mysql_real_connect(&mysql, host, user, passwd, database, 0, NULL, 0)){
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
mysql_error(&mysql));
    return 0;
}
fprintf(stdout, "Listen %s start!!\n", argv[1]);

while(!maxk || (ipak < maxk || maxk==0))
{

```



```

    frmlen = recv(listen_fd,buffer,192,MSG_TRUNC); //0->flags
(MSG_PEEK,MSG_OOB,MSG_WAITALL,MSG_TRUNC)

    if(frmlen < 32)
        continue;

    eptr = (struct ether_header *) buffer;
    ether_type = ntohs(eptr->ether_type);
    if(ether_type != ETHERTYPE_IP)
        continue;

    // get the address of protocols
    ip = (struct iphdr *)(buffer + sizeof(struct ethhdr));
    if(ip->ihl==7)
    {
        top = (struct Tempop *)(buffer + sizeof(struct ethhdr) + sizeof(struct iphdr));
        if(top->option != 27)
            continue;

        /* ##### packet record start
##### */
        Opbuf = (OPs *) malloc(sizeof(struct ops));

        // get the time and date
        time(&t);

        memcpy(&(Opbuf->Ts),&t, sizeof(time_t));
        memcpy(&(Opbuf->Te),&t, sizeof(time_t));

        // TCP
        if(ip->protocol==6)
            tcp = (struct tcphdr *)(buffer + sizeof(struct ethhdr) + sizeof(struct
iphdr) + sizeof(struct Tempop) );
        // UDP
        else if(ip->protocol==17)
            udp = (struct udphdr *)(buffer + sizeof(struct ethhdr) + sizeof(struct
iphdr) + sizeof(struct Tempop) );
        else
            continue;

        // source IP and destination IP
        Opbuf->source_IP = *(int *)&ip->saddr;
        Opbuf->source_IP = (Opbuf->source_IP>>24 & 0xFF) |
(Opbuf->source_IP>>8 & 0xFF00) | (Opbuf->source_IP<<8 & 0xFF0000) |
(Opbuf->source_IP<<24 & 0xFF000000);
        Opbuf->dest_IP = *(int *)&ip->daddr;
        Opbuf->dest_IP = (Opbuf->dest_IP>>24 & 0xFF) | (Opbuf->dest_IP>>8 &
0xFF00) | (Opbuf->dest_IP<<8 & 0xFF0000) | (Opbuf->dest_IP<<24 & 0xFF000000);

        // IP protocol

```

```

Opbuf->protocol = ip->protocol;

if(ip->protocol==6)
{
    // TCP - source port & destination port
    Opbuf->source_PORT = ntohs(tcp->source);
    Opbuf->dest_PORT = ntohs(tcp->dest);
}
else if(ip->protocol==17)
{
    // UDP - source port & destination port
    Opbuf->source_PORT = ntohs(udp->source);
    Opbuf->dest_PORT = ntohs(udp->dest);
}

// Packet option
Opbuf->op.option = top->option;
Opbuf->op.length = top->length;
Opbuf->op.hash = (top->ops[4]&0xF) | top->ops[5];
Opbuf->op.IID[0]= top->ops[0] | ((top->ops[4]&0x80)<<1);
Opbuf->op.IID[1]= top->ops[1] | ((top->ops[4]&0x40)<<2);
Opbuf->op.IID[2]= top->ops[2] | ((top->ops[4]&0x20)<<3);
Opbuf->op.IID[3]= top->ops[3] | ((top->ops[4]&0x10)<<4);
Opbuf->op.IID[4]= 0;
Opbuf->op.IID[5]= 0;

// IID number
Opbuf->IID_Num = 0;
if(Opbuf->op.IID[Opbuf->IID_Num]==0)
    continue;
while(Opbuf->op.IID[Opbuf->IID_Num]!=0)
{
    Opbuf->IID_Num ++;
}
if(Opbuf->op.IID[Opbuf->IID_Num-1]!=IID){
    Opbuf->op.IID[Opbuf->IID_Num] = IID;
    Opbuf->IID_Num++;
}else{
    if(Opbuf->IID_Num != 1)
        continue;
}

// pointer default
Opbuf->next = NULL;
Opbuf->pre = NULL;
Opbuf->IID_next = NULL;
Opbuf->IID_pre = NULL;

/* ##### packet record end
##### */

```



```

sigprocmask(SIG_BLOCK,&intmask,NULL);
// search the buffer whether the record is exist
if(IIDListH[Opbuf->IID_Num-1] ==NULL)
{
    // The Head is NULL
    IIDListH[Opbuf->IID_Num-1] = Opbuf;
    IIDListT[Opbuf->IID_Num-1] = Opbuf;
    if(TimeListH == NULL)
    {
        TimeListH = Opbuf;
        TimeListT = Opbuf;
    }else{
        TimeListT->next = Opbuf;
        Opbuf -> pre = TimeListT;
        TimeListT = Opbuf;
    }
    RecordNum++;

    if(RecordNum > MAX_BUFFER)
    {
        recordMAX();
    }
    //printPacket(Opbuf);////////// print the packet information
}else{
    // search buffer other than head
    tbuf = IIDListH[Opbuf->IID_Num-1];
    while(tbuf !=NULL)
    {
        if(compareBuf(tbuf,Opbuf))
            break;
        tbuf = tbuf->IID_next;
    }

    if(tbuf != NULL)
    {
        memcpy(&(tbuf->Te), &(Opbuf->Te), sizeof(time_t));
        free(Opbuf);
    }else{
        // compare not found
        IIDListT[Opbuf->IID_Num-1]->IID_next = Opbuf;
        Opbuf->IID_pre = IIDListT[Opbuf->IID_Num-1];
        IIDListT[Opbuf->IID_Num-1] = Opbuf;

        TimeListT ->next = Opbuf;
        Opbuf->pre = TimeListT;
        TimeListT = Opbuf;

        RecordNum++;
        if(RecordNum >MAX_BUFFER)

```

```
        {
            recordMAX();
        }
        //printPacket(Opbuf);////////
    }
}
sigprocmask(SIG_UNBLOCK,&intmask,NULL);
}else{
    continue;
}

    ipak++;
}

mysql_close(&mysql);
return 0;

FAIL:
close(listen_fd);
return -1;
}
```



Code 3 myd.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <mysql/mysql.h>
#define MAX_DAY 7

int main(int argc, char **argv)
{
    time_t t; // time
    struct tm sts;
    char host[] = "127.0.0.1";
    char database[] = "wnl";
    char user[] = "root";
    char passwd[] = "wnl";
    char *query;
    MYSQL mysql;

    mysql_init(&mysql);

    // get the time and date
    t = time(NULL) - MAX_DAY * 3600 * 24;

    #ifdef SunOS
        memcpy(&sts, localtime(&t), sizeof(struct tm));
    #else
        localtime_r(&t, &sts);
    #endif

    if(!mysql_real_connect(&mysql, host, user, passwd, database, 0, NULL, 0)){
        fprintf(stderr, "Failed to connect to database: Error: %s\n",
mysql_error(&mysql));
        return 0;
    }

    query = (char *) malloc(256*sizeof(char));
    sprintf(query, "delete from tam where STime<='%04d-%02d-%02d
%02d:%02d:%02d'", sts.tm_year+1900, sts.tm_mon+1, sts.tm_mday, sts.tm_hour, sts.tm_min,
sts.tm_sec);
    printf("%s", query);

    if(mysql_real_query(&mysql, query, strlen(query))){
        fprintf(stderr, "Failed to update database: Error: %s\n", mysql_error(&mysql));
    }

    mysql_close(&mysql);
}
```

```
return 0;  
}
```



Code 4 **Traceback.java**

```
import javax.swing.*;
import java.awt.*;
import java.util.*;
import java.net.*;
import java.awt.event.*;
import java.text.SimpleDateFormat;
import TBapps.*;

public class Traceback extends JFrame implements ItemListener, ActionListener
{

    JPanel TBPanel;
    JPanel TBPanel_Row1 = new JPanel(new FlowLayout(FlowLayout.LEFT));
    JPanel TBPanel_Row2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
    JPanel TBPanel_Row3 = new JPanel(new FlowLayout(FlowLayout.LEFT));

    JTextField SourceText;
    JTextField DestPortText;
    JTextField ipprotocolText;
    JTextField Search_IP_Text;
    JTextField DestText;
    JTextField RouterText;
    JTextField TimeText;
    JTextField DateText;

    JComboBox yearBox = new JComboBox();
    JComboBox monthBox = new JComboBox();
    JComboBox dayBox = new JComboBox();
    JComboBox hourBox = new JComboBox();
    JComboBox minuteBox = new JComboBox();
    JComboBox secondBox = new JComboBox();
    JComboBox differBox = new JComboBox();

    JCheckBox time_checkbox = new JCheckBox();
    JCheckBox date_checkbox = new JCheckBox();

    JButton searchpathButton;

    GregorianCalendar day;

    // another java code
    SimpleTable Stable;

    Traceback()
    {
        // 離開視窗的監聽
        addWindowListener(
            new WindowAdapter()
```

```

        {
            public void windowClosing(WindowEvent we)
            {
                System.exit(0);
            }
        }
    );

    // 最底層 Panel
    TPanel = new JPanel();
    TPanel.setLayout(new GridLayout(4, 1));

    // 查詢欄位設定
    SourceText = new JTextField("?", 10);
    DestText = new JTextField("?", 10);
    DestPortText = new JTextField("?", 5);
    ipprotocolText = new JTextField("?", 6);
    RouterText = new JTextField("127.0.0.1", 10);
    TimeText = new JTextField("?", 10);
    DateText = new JTextField("?", 10);
    searchpathButton = new JButton("Search");

    Stable = new SimpleTable();

    // 日期時間宣告
    day = new GregorianCalendar();

    //開始加入年
    for(int i=day.get(Calendar.YEAR)-8; i<=day.get(Calendar.YEAR); i++)
    {
        yearBox.addItem(i);
    }
    yearBox.setSelectedIndex(8);

    //開始加入月
    for(int i=1; i<=12; i++)
    {
        monthBox.addItem(i);
    }
    monthBox.setSelectedIndex(day.get(Calendar.MONTH));

    //開始加入日期
    for(int i=1; i<=31; i++)
    {
        dayBox.addItem(i);
    }
    dayBox.setSelectedIndex(day.get(Calendar.DATE)-1);

    //開始加入時

```

```

for(int i=0; i<=23; i++)
{
    hourBox.addItem(i);
}
hourBox.setSelectedIndex(day.get(Calendar.HOUR_OF_DAY));

//開始加入分
for(int i=0; i<=59; i++)
{
    minuteBox.addItem(i);
}
minuteBox.setSelectedIndex(day.get(Calendar.MINUTE));

// 開始加入秒
for(int i=0; i<=59; i++)
{
    secondBox.addItem(i);
}
secondBox.setSelectedIndex(day.get(Calendar.SECOND));

// 正負時間差
for(int i=1; i<=60; i++)
{
    differBox.addItem(i);
}

// 初始化時間與日期欄位
TimeText.setEditable(false);
DateText.setEditable(false);

```

```

TimeText.setText(hourBox.getSelectedItem().toString()+":"+minuteBox.getSelectedItem().toString()+":"+secondBox.getSelectedItem().toString());

```

```

DateText.setText(yearBox.getSelectedItem().toString()+"-"+monthBox.getSelectedItem().toString()+"-"+dayBox.getSelectedItem().toString());

```

```

// 加入 row 1
TBPannel.add(TBPannel_Row1);
TBPannel_Row1.add(new JLabel("Source IP:"));
TBPannel_Row1.add(SourceText);
TBPannel_Row1.add(new JLabel("DestIP:"));
TBPannel_Row1.add(DestText);
TBPannel_Row1.add(new JLabel("DestPort:"));
TBPannel_Row1.add(DestPortText);
TBPannel_Row1.add(new JLabel("IP Protocol:"));
TBPannel_Row1.add(ipprotocolText);
TBPannel_Row1.add(new JLabel("Router IP : "));
TBPannel_Row1.add(RouterText);

```

```

// 加入 row 2
TbPanel.add(TbPanel_Row2);
TbPanel_Row2.add(date_checkbox);
TbPanel_Row2.add(new JLabel("Date : "));
TbPanel_Row2.add(DateText);
TbPanel_Row2.add(time_checkbox);
TbPanel_Row2.add(new JLabel("Time : "));
TbPanel_Row2.add(TimeText);
TbPanel_Row2.add(yearBox);
TbPanel_Row2.add(new JLabel("年"));
TbPanel_Row2.add(monthBox);
TbPanel_Row2.add(new JLabel("月"));
TbPanel_Row2.add(dayBox);
TbPanel_Row2.add(new JLabel("日"));
TbPanel_Row2.add(hourBox);
TbPanel_Row2.add(new JLabel("時"));
TbPanel_Row2.add(minuteBox);
TbPanel_Row2.add(new JLabel("分"));
TbPanel_Row2.add(secondBox);
TbPanel_Row2.add(new JLabel("秒"));

// 加入 row 3
TbPanel.add(TbPanel_Row3);
TbPanel_Row3.add(new JLabel("正負"));
TbPanel_Row3.add(differBox);
TbPanel_Row3.add(new JLabel("分區間均列出"));

// 欄位變更監聽
yearBox.addItemListener(this);
monthBox.addItemListener(this);
dayBox.addItemListener(this);
hourBox.addItemListener(this);
minuteBox.addItemListener(this);
secondBox.addItemListener(this);
differBox.addItemListener(this);

// 按鈕監聽
searchpathButton.addActionListener(this);

// 視窗設定
getContentPane().add(BorderLayout.NORTH, TbPanel);
getContentPane().add(BorderLayout.CENTER, Stable);
getContentPane().add(BorderLayout.SOUTH, searchpathButton);

setVisible(true);
setSize(1024, 768);
}

```



```

public void itemStateChanged(ItemEvent ie)
{
    // 改變 日期時間 內容
    String year_str = yearBox.getSelectedItem().toString();
    String month_str = monthBox.getSelectedItem().toString();
    String day_str = dayBox.getSelectedItem().toString();
    String hour_str = hourBox.getSelectedItem().toString();
    String minute_str = minuteBox.getSelectedItem().toString();
    String second_str = secondBox.getSelectedItem().toString();

    String time_str = hour_str + ":" + minute_str + ":" + second_str;
    String date_str = year_str + "-" + month_str + "-" + day_str;
    TimeText.setText(time_str);
    DateText.setText(date_str);

}

public void actionPerformed(ActionEvent e)
{
    if (e.getSource().equals(searchpathButton))
    {
        //按下 search button 的執行
        long SsIP=0;
        long DdIP=0;
        String[] records;
        String[] records_cols;
        String whereStr;

        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
        Calendar SCal=Calendar.getInstance(),ECal=Calendar.getInstance();

        String sql="SELECT * FROM tam";

        String[] siplist = (SourceText.getText()).split("[.]");
        if(siplist.length==4)
            SsIP = Long.parseLong(siplist[0])<<24 | Long.parseLong(siplist[1])<<16
| Long.parseLong(siplist[2])<<8 | Long.parseLong(siplist[3]);

        String[] diplist = (DestText.getText()).split("[.]");
        if(diplist.length==4)
            DdIP = Long.parseLong(diplist[0])<<24 |
Long.parseLong(diplist[1])<<16 | Long.parseLong(diplist[2])<<8 |
Long.parseLong(diplist[3]);

        whereStr = "";
        // source IP

```

```

if(!SourceText.getText().equals("*"))
{
    if(whereStr == "")
        whereStr = " where";
    else
        whereStr += " and";
    whereStr += " SIP=" + Long.toString(SsIP);
}

// dest IP
if(!DestText.getText().equals("*"))
{
    if(whereStr == "")
        whereStr = " where";
    else
        whereStr += " and";
    whereStr += " DIP=" + Long.toString(DdIP);
}

// dest port
if(!DestPortText.getText().equals("*"))
{
    if(whereStr == "")
        whereStr = " where";
    else
        whereStr += " and";
    whereStr += " DPORT=" + DestPortText.getText();
}

// ip porotocl
if(!ipprotocolText.getText().equals("*"))
{
    if(whereStr == "")
        whereStr = " where";
    else
        whereStr += " and";
    whereStr += " Protocol=" + ipprotocolText.getText();
}

// Time
if(date_checkbox.isSelected()){
    Integer differ_time =
Integer.parseInt(differBox.getSelectedItem().toString());
    Integer Myear = Integer.parseInt(yearBox.getSelectedItem().toString());
    Integer Mmonth =
Integer.parseInt(monthBox.getSelectedItem().toString())-1;
    Integer Mday = Integer.parseInt(dayBox.getSelectedItem().toString());

    Integer Mhour = Integer.parseInt(hourBox.getSelectedItem().toString());
    Integer Mminute =

```

```

Integer.parseInt(minuteBox.getSelectedItem().toString());
    Integer Msecond =
Integer.parseInt(secondBox.getSelectedItem().toString());
    if(!time_checkbox.isSelected()){
        Mhour = 0;
        Mminute = 0;
        Msecond = 0;
    }

    SCal.set(Myear,Mmonth,Mday,Mhour,Mminute,Msecond);
    ECal.set(Myear,Mmonth,Mday,Mhour,Mminute,Msecond);

    if(time_checkbox.isSelected()){
        SCal.add(Calendar.MINUTE, -differ_time);
        ECal.add(Calendar.MINUTE, differ_time);
    }else{
        ECal.add(Calendar.HOUR,24);
    }

    if(whereStr == "")
        whereStr = " where";
    else
        whereStr += " and";
    whereStr += " STime<=" + formatter.format(EScal.getTime()) + " and
ETime>=" + formatter.format(SCal.getTime()) + """;
    }

    sql += whereStr;

    // 進行查詢的動作 connection action
    MessagePacket currPacket = null;
    Integer usePort = 4862;
    String recv_string;
    try {
        ClientThread client = new
ClientThread(InetAddress.getByAddress(RouterText.getText()), usePort);
        client.send_message(new MessagePacket("traceback", sql));

        MessagePacket revPacket = client.receive_message();

        if(revPacket.getType().equals("tracebackEop")) {
            Stable.removeAllRows();
            recv_string = revPacket.toString();
            records = recv_string.split("\n");
            for(int i=0;i<records.length;i++){
                records_cols = records[i].split("/");
                Stable.myinsert(records_cols);
            }
        }
    }

```

```
        }
        client.close_connection();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

public static void main(String[] args)
{
    Traceback tb = new Traceback();
}
}
```



Code 5 br_forward.c

```
/*
 * Forwarding decision
 * Linux ethernet bridge
 *
 * Authors:
 * Lennert Buytenhek <buytenh@gnu.org>
 *
 * $Id: br_forward.c,v 1.4 2001/08/14 22:05:57 davem Exp $
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version
 * 2 of the License, or (at your option) any later version.
 *
 * Change log: br_forward.c, modified by Tim Hann Huang 2009/07/06
 */

#include <linux/kernel.h>
#include <linux/netdevice.h>
#include <linux/skbuff.h>
#include <linux/if_vlan.h>
#include <linux/netfilter_bridge.h>
#include "br_private.h"
#include <linux/ip.h>
#include <net/ip.h>
#include <linux/unistd.h>

#include "md5.h"

#define ETHERTYPE_IP 0x0800

extern unsigned int MYIID;
extern unsigned int MYRIP[30];
extern unsigned int MYSUB[30];
extern unsigned int MYCNT;
extern unsigned int SETPN;

/* Don't forward packets to originating port or forwarding disabled */
static inline int should_deliver(const struct net_bridge_port *p,
                                const struct sk_buff *skb)
{
    return (skb->dev != p->dev && p->state == BR_STATE_FORWARDING);
}

static inline unsigned packet_length(const struct sk_buff *skb)
{
    return skb->len - (skb->protocol == htons(ETH_P_8021Q) ? VLAN_HLEN : 0);
}
```



```

int br_dev_queue_push_xmit(struct sk_buff *skb)
{
    /* drop mtu oversized packets except gso */
    if (packet_length(skb) > skb->dev->mtu && !skb_is_gso(skb))
        kfree_skb(skb);
    else {
        /* ip_refrag calls ip_fragment, doesn't copy the MAC header. */
        if (nf_bridge_maybe_copy_header(skb))
            kfree_skb(skb);
        else {
            skb_push(skb, ETH_HLEN);

            dev_queue_xmit(skb);
        }
    }

    return 0;
}

int br_forward_finish(struct sk_buff *skb)
{
    return NF_HOOK(PF_BRIDGE, NF_BR_POST_ROUTING, skb, NULL, skb->dev,
        br_dev_queue_push_xmit);
}

static void __br_deliver(const struct net_bridge_port *to, struct sk_buff *skb)
{
    skb->dev = to->dev;
    NF_HOOK(PF_BRIDGE, NF_BR_LOCAL_OUT, skb, NULL, skb->dev,
        br_forward_finish);
}

static void __br_forward(const struct net_bridge_port *to, struct sk_buff *skb)
{
    struct net_device *indev;

    // add my thing
    unsigned int i,j;
    struct ethhdr *ethh;
    struct iphdr *iph;
    unsigned char *optptr;
    unsigned int hashes;
    char buffer[4];
    md5_t md5;
    unsigned char sig[MD5_SIZE];

    // ***

```

```

indev = skb->dev;
skb->dev = to->dev;
skb_forward_csum(skb);

// ***
ethh = eth_hdr(skb);
if(ntohs(ethh->h_proto) == ETHERTYPE_IP){
    iph = ip_hdr(skb);

    i=1;
    j=0;
    while(i<=MYCNT){
        if (!(iph->daddr ^ MYRIP[i-1] & MYSUB[i-1])){
            j=1;
            break;
        }
        i++;
    }

    if(j==SETPN){
        if((iph->ihl==5) && (ntohs(iph->tot_len)<1488) && ((iph->protocol)==6 ||
(iph->protocol)==17)) {
            iph->ihl += 2;
            iph->tot_len = htons(ntohs(iph->tot_len) + 8);
            skb_push(skb,8); // 8 byte

            memcpy((unsigned char *)ethh-8, (unsigned char *)ethh, sizeof(struct
ethhdr));
            memcpy((unsigned char *)iph-8, (unsigned char *)iph, sizeof(struct
iphdr));
            optptr = (unsigned char *)ethh + sizeof(struct ethhdr) + sizeof(struct
iphdr) - 8;

            iph = (struct iphdr *)((unsigned char *)iph - 8);

            hash = 0;
            *optptr = 27; //option = 27;
            optptr[1] = 0x08; // unit 1 byte (length)
            optptr[2] = MYIID & 0xFF; // IID1:8 bits
            optptr[3] = 0x0; // IID2:8 bits
            optptr[4] = 0x0; // IID3:8 bits
            optptr[5] = 0x0; // IID4:8 bits
            optptr[6] = (MYIID & 0x100)>>1 | ((hash & 0xF00)>>8); //
IID1:1bit,IID2:1bit,IID3:1bit,IID4:1bit,HASH:4bits

            buffer[0] = (((unsigned int)(iph->saddr)>>8) & 0xFF) ^ optptr[5] ^
(optptr[6]&0xF0);
            buffer[1] = ((iph->daddr>>8) & 0xFF) ^ optptr[4] ^ (iph->id & 0xFF);
            buffer[2] = ((iph->saddr>>24) & 0xFF) ^ optptr[3] ^ (optptr[6]&0xF0);
            buffer[3] = ((iph->daddr>>24) & 0xFF) ^ optptr[2] ^ ((iph->id>>8) &

```

```

0xFF);

        md5_init(&md5);
        md5_process(&md5, buffer, 4);
        md5_finish(&md5, sig);

        hashes = (sig[0] ^ sig[1] ^ sig[6] ^ sig[7]) | ((sig[3] ^ sig[4] ^ sig[9] ^
sig[10])<<8);
        optptr[6] = (MYIID & 0x100)>>1 | ((hashs & 0x0F00)>>8); //
IID1:1bit,IID2:1bit,IID3:1bit,IID4:1bit,HASH:4bits
        optptr[7] = hashs & 0xFF; // HASH:8 bits

        ip_send_check(iph);
    }else{
        if(iph->ihl==7){
            optptr = (unsigned char *)iph + sizeof(struct iphdr);
            if(*optptr == 27){

                buffer[0] = (((unsigned int)(iph->saddr))>>8) & 0xFF) ^
optptr[5] ^ (optptr[6]&0xF0);
                buffer[1] = ((iph->daddr)>>8) & 0xFF) ^ optptr[4] ^ (iph->id
& 0xFF);
                buffer[2] = ((iph->saddr)>>24) & 0xFF) ^ optptr[3] ^
(optptr[6]&0xF0);
                buffer[3] = ((iph->daddr)>>24) & 0xFF) ^ optptr[2] ^
((iph->id)>>8) & 0xFF);

                md5_init(&md5);
                md5_process(&md5, buffer, 4);
                md5_finish(&md5, sig);

                hashes = (sig[0] ^ sig[1] ^ sig[6] ^ sig[7]) | ((sig[3] ^ sig[4] ^
sig[9] ^ sig[10])<<8);

                if((hashs&0x0FFF) != (((optptr[6]&0x0F)<<8)|optptr[7])){
                    optptr[0] = 0;
                }else{
                    if(optptr[3]==0 && (optptr[6]&0x40)==0){
                        optptr[3]=MYIID & 0xFF;
                        optptr[6]=optptr[6] | ((MYIID & 0x100)>>2);
                    }else if(optptr[4]==0 && (optptr[6]&0x20)==0){
                        optptr[4]=MYIID & 0xFF;
                        optptr[6]=optptr[6] | ((MYIID & 0x100)>>3);
                    }else if(optptr[5]==0 && (optptr[6]&0x10)==0){
                        optptr[5]=MYIID & 0xFF;
                        optptr[6]=optptr[6] | ((MYIID & 0x100)>>4);
                    }else{
                        optptr[4]=optptr[5];
                        optptr[5]=MYIID & 0xFF;
                        optptr[6]=(optptr[6] & 0xCF) |

```



```

((optptr[6]&0x10)<<1) | ((MYIID&0x100)>>4);
        }
    }
    ip_send_check(ip);
}
}
}
} else {
    if(ip->ihl==7 && ((ip->protocol)==6 || (ip->protocol)==17)) {
        optptr = (unsigned char *)iph + sizeof(struct iphdr);
        if(*optptr == 27) {
            ip->ihl -= 2;
            ip->tot_len = htons(ntohs(ip->tot_len) - 8);

            memcpy((unsigned char *)iph+20,(unsigned char *)iph+12,8);
            memcpy((unsigned char *)iph+12,(unsigned char *)iph+4,8);
            memcpy((unsigned char *)iph+8,(unsigned char *)iph,4);
            memcpy((unsigned char *)ethh+14,(unsigned char *)ethh+6,8);
            memcpy((unsigned char *)ethh+8,(unsigned char *)ethh,6);
            skb_pull(skb,8);
            ip = (struct iphdr *)((unsigned char *)iph + 8);
            ip_send_check(ip);
        }
    }
}
}
// end of add
}

NF_HOOK(PF_BRIDGE, NF_BR_FORWARD, skb, indev, skb->dev,
        br_forward_finish);
}

/* called with rcu_read_lock */
void br_deliver(const struct net_bridge_port *to, struct sk_buff *skb)
{
    if (should_deliver(to, skb)) {
        __br_deliver(to, skb);
        return;
    }

    kfree_skb(skb);
}

/* called with rcu_read_lock */
void br_forward(const struct net_bridge_port *to, struct sk_buff *skb)
{
    if (should_deliver(to, skb)) {
        __br_forward(to, skb);
        return;

```



```

    }

    kfree_skb(skb);
}

/* called under bridge lock */
static void br_flood(struct net_bridge *br, struct sk_buff *skb,
    void (*__packet_hook)(const struct net_bridge_port *p,
        struct sk_buff *skb))
{
    struct net_bridge_port *p;
    struct net_bridge_port *prev;

    prev = NULL;

    list_for_each_entry_rcu(p, &br->port_list, list) {
        if (should_deliver(p, skb)) {
            if (prev != NULL) {
                struct sk_buff *skb2;

                if ((skb2 = skb_clone(skb, GFP_ATOMIC)) == NULL) {
                    br->statistics.tx_dropped++;
                    kfree_skb(skb);
                    return;
                }

                __packet_hook(prev, skb2);
            }

            prev = p;
        }
    }

    if (prev != NULL) {
        __packet_hook(prev, skb);
        return;
    }

    kfree_skb(skb);
}

/* called with rcu_read_lock */
void br_flood_deliver(struct net_bridge *br, struct sk_buff *skb)
{
    br_flood(br, skb, __br_deliver);
}

/* called under bridge lock */
void br_flood_forward(struct net_bridge *br, struct sk_buff *skb)

```

```
{  
    br_flood(br, skb, __br_forward);  
}
```



Appendix B. Setting Procedures of an IPM Router

Here is the setting of an IPM router. The following description is the steps of installing and controlling an IPM router.

At first, we download the Ubuntu 8.04 LTS ISO image and install to a computer. The guide will teach us, step-by-step, to finish the installation. We change Linux kernel to our modified version so that we can use our commands to revise variables inside the kernel.

We open command window to install packets which we need from Internet. Figure 28 depicts the installation of packets from Internet. After the installation of these commands, we already have bridge modules, a MySQL database and java environment.



```
root@wnl-PC1: ~
檔案(E) 編輯(E) 顯示(V) 終端機(T) 分頁(B) 求助(H)
root@wnl-PC1:~# apt-get install g++ bridge-utils mysql-server-5.0 libmysqlclient
15-dev phpmyadmin libmysql-java sun-java6-jdk
讀取套件清單中... 完成
了解套件依存關係中
Reading state information... 完成
『bridge-utils』已經是最新版本了。
『mysql-server-5.0』已經是最新版本了。
『libmysqlclient15-dev』已經是最新版本了。
『phpmyadmin』已經是最新版本了。
『libmysql-java』已經是最新版本了。
『sun-java6-jdk』已經是最新版本了。
下列的【新】套件都將被安裝：
  g++-4.2 libstdc++6-4.2-dev
建議(Suggested)的套件：
  g++-multilib g++-4.2-multilib gcc-4.2-doc libstdc++6-4.2-dbg
  libstdc++6-4.2-doc
下列的【新】套件都將被安裝：
  g++ g++-4.2 libstdc++6-4.2-dev
更新 0 個套件，新安裝 3 個套件，刪除 0 個套件，另不更新 0 個套件。
需要下載 3973kB 的檔案。
After this operation, 15.0MB of additional disk space will be used.
繼續執行嗎？ 是按 [Y] 鍵，否按 [n] 鍵 y
```

Figure 28 Installation of packets from Internet

The next step is setting the configuration of the database. We add an administrator, a database name and a table into the database. Figure 29 depicts the table in the database. We can see the setting of each fields and types of them.

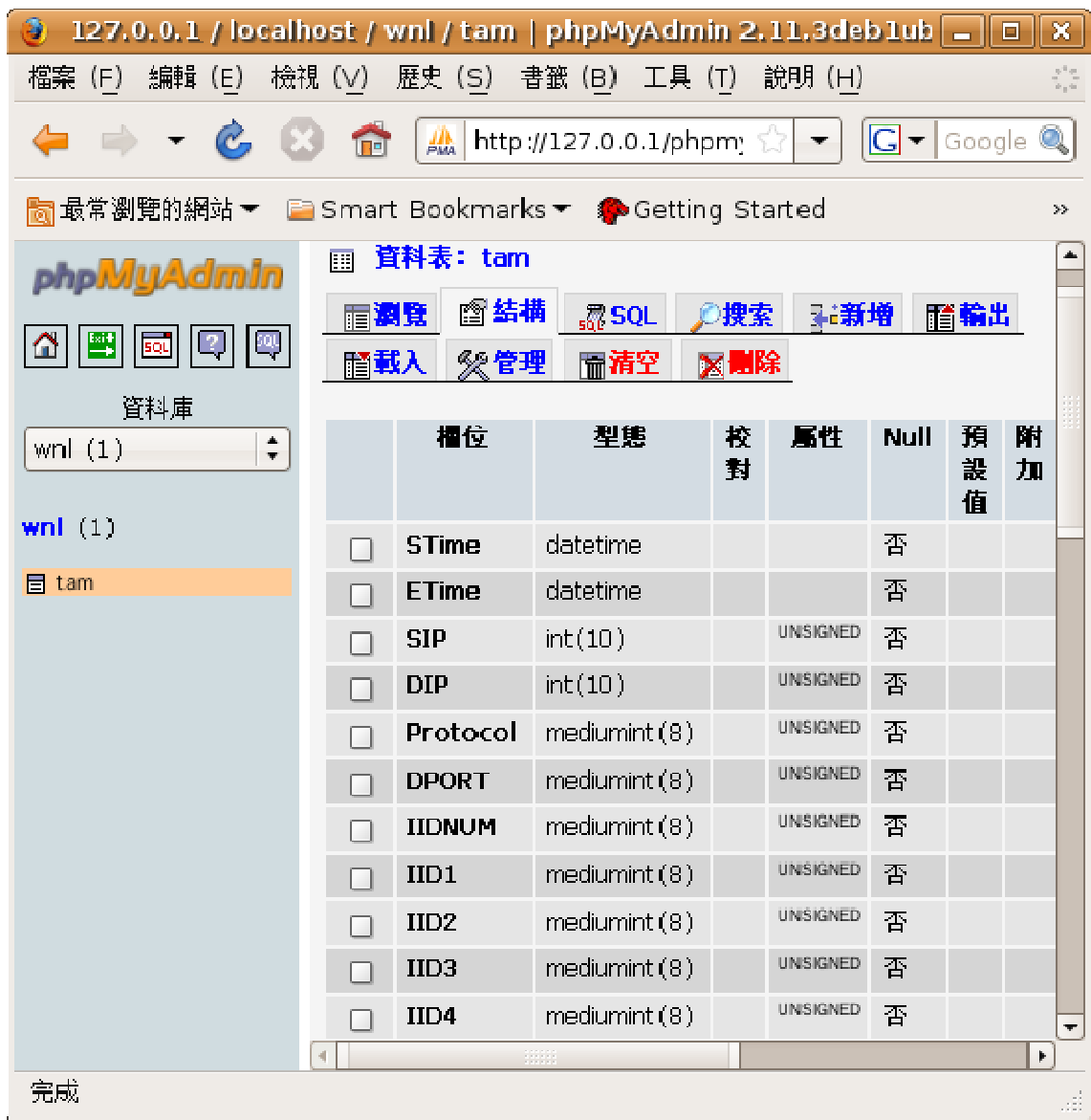


Figure 29 The table “tam” in the database “wnl”

The following step is the setup of changing a PC into a bridge. We install the bridge-utils packets for the modules of bridge. We still need to start it up so that it could work for a PC. Figure 30 depicts how to add a bridge name “br0” and add interface into the bridge. The bridge will start after the command “ifconfig br0 up”.

```

root@wnl-PC1: ~
檔案(E) 編輯(E) 顯示(V) 終端機(T) 分頁(B) 求助(H)
root@wnl-PC1:~# brctl addbr br0
root@wnl-PC1:~# brctl addif br0 eth0
root@wnl-PC1:~# brctl addif br0 eth1
root@wnl-PC1:~# brctl show
bridge name      bridge id                STP enabled      interfaces
br0              8000.001fd087342c       no               eth0
                                                         eth1

root@wnl-PC1:~# ifconfig br0 up
root@wnl-PC1:~# ifconfig br0
br0              Link encap:Ethernet  HWaddr 00:1f:d0:87:34:2c
                inet6 addr: fe80::21f:d0ff:fe87:342c/64 Scope:Link
                UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
                RX packets:0 errors:0 dropped:0 overruns:0 frame:0
                TX packets:5 errors:0 dropped:0 overruns:0 carrier:0
                collisions:0 txqueuelen:0
                RX bytes:0 (0.0 B)  TX bytes:398 (398.0 B)

root@wnl-PC1:~# █

```

Figure 30 Commands for bridge setup

We have to decide the IID of the PC. The IID value will modify by the program written by us. We add network rule for marking packets. The control is shown in Figure 31. We set IID into 16. Network rules are set to the kernel. The packets through these two networks will be marked.

```

root@wnl-PC1: ~
檔案(E) 編輯(E) 顯示(V) 終端機(T) 分頁(B) 求助(H)
root@wnl-PC1:~# ./mpcset.o iid 16
IID = 16
root@wnl-PC1:~# ./mpcset.o rip add 192.168.1.0/24
Add the record into RIP!
IP:192.168.1.0 submask:FFFFFF00
root@wnl-PC1:~# ./mpcset.o rip add 192.168.2.0/24
Add the record into RIP!
IP:192.168.2.0 submask:FFFFFF00
root@wnl-PC1:~# ./mpcset.o rip show
The records(Total:2):
IP:192.168.1.0 submask:FFFFFF00
IP:192.168.2.0 submask:FFFFFF00
root@wnl-PC1:~# ./mpcset.o setpn 0
SETPN = 0
root@wnl-PC1:~# █

```

Figure 31 The setting of network rules

We have to execute sniffer for listening to the packets with marking information. The

sniffers are executed by the commands shown in Figure 32.



Figure 32 Sniffer for each interface

The last step is opening the service for traceback. We execute a program for listening to the port 4862. Therefore, PC would response the result of searching the database to the source of packets. Figure 33 depicts that server program is executed by java.



Figure 33 Server on the IPM router

An IPM router is complete after the setting and installation. We could put it into the network which we want to mark packets.