AUTHENTICATION NETWORK ACCESS CONTROL BY USING IPV4

PARINYA THAMTHAWORNSAKUL

A THESIS SUBMITTED IN FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF ENGINEERING IN TELECOMMUNICATIONS ENGINEERING
FACULTY OF ENGINEERING
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2018

KMITL-2018-EN-M-010-173

AUTHENTICATION NETWORK ACCESS CONTROL BY USING IPV4

PARINYA THAMTHAWORNSAKUL

A THESIS SUBMITTED IN FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN TELECOMMUNICATIONS ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2018

KMITL-2018-EN-M-010-173

การตรวจสอบการลงทะเบียนเพื่อเข้าใช้งานโครงข่าย IPV4

AUTHENTICATION NETWORK ACCESS CONTROL BY USING IPV4

ปริญญา ธรรมถาวรสกุล

PARINYA THAMTHAWORNSAKUL

วิทยานิพนธ์นี้สำหรับการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ. 2561

KMITL-2018-EN-M-010-173

หัวข้อวิทยานิพนธ์	การตรวจสอบการลงทะเบียนเพื่อเข้าใช้งานโครงข่าย IPV4
นักศึกษา	นายปริญญา ธรรมถาวรสกุล
รหัสประจำตัว	56601420
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมโทรคมนาคม
พ.ศ.	2561
อาจารย์ที่ปรึกษาวิทยานิพนธ์	รศ.ดร.สุวิพล สิทธิชีวภาค

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้ นำเสนอการควบคุมการเข้าถึงเครือข่ายที่มีการรับรองความถูกต้องแบบไม่มีช่วงเวลา ซึ่งเป็นการเพิ่มความสามารถการรับรองความถูกต้องด้วยตัวเองให้กับมาตรฐานการสื่อสารอินเตอร์เน็ตโปรโตคอลรุ่นที่ 4 โดยตัวเลือกของอินเตอร์เน็ตโปรโตคอลรุ่นที่ 4 ที่ยังไม่มีการใช้งานจะถูกนำมากำหนดเพื่อใช้บรรจุข้อมูลที่สำคัญเพื่อวัตถุประสงค์ในการระบุตัวผู้ใช้งาน การรับรองข้อมูลในแพ็กเกต และการรักษาความปลอดภัย ส่วนประกอบสำคัญที่ทำให้กลไกของการรับรองความถูกต้องสามารถทำได้โดยไม่ต้องมีช่วงเวลาคือ HMAC (Keyed-Hashing for Message Authentication) การนำการรับรองความถูกต้องแบบไม่มีช่วงเวลาไปใช้งาน จะทำให้แพ็กเกตอินเตอร์เน็ตโปรโตคอลรุ่นที่ 4 มีคุณสมบัติที่สามารถตรวจสอบความสมบูรณ์ของข้อมูล และรับรองแหล่งที่มาของข้อมูลได้ ประโยชน์ที่ได้คือข้อมูลบนเครือข่ายสามารถถูกตรวจสอบความเป็นต้นฉบับได้ ทำให้มีความปลอดภัยจากการโจมตีแบบ MITM (Man-In-The-Middle) และการระบุตัวผู้ใช้งานมีความแม่นยำมากขึ้น

Thesis	Authentication Network Access Control by Using IPV4
Student	Mr. Parinya Thamthawornsakul
Student ID.	56601420
Degree	Master of Engineering
Program	Telecommunications Engineering
Year	2018
Thesis Advisor	Assoc. Prof. Dr. Suvepon Sittichivapak

ABSTRACT

This thesis proposes a sessionless authentication network access control which provides ability of self-authentication to the standard IPv4 communication. Unspecified IPv4 options are defined and used to carry important information for the purposes of user identification, packet authentication, and security. The HMAC (Keyed-Hashing for Message Authentication) is a key component that makes authentication mechanism become sessionless. Application of sessionless authentication will provide data integrity verification and data origin authentication features to IPv4 packets. The significant benefits are information transmitting over the IPv4 network can be checked for originality which will be safe from MITM (Man-In-The-Middle) attack and user identification will be more accurate.

ACKNOWLEDGEMENT

I would like to express my deepest and sincere gratitude to my thesis advisor, Assoc.Prof.Dr.Suvepon Sittichivapak for his kindness in providing an opportunity to be his advisee. I am also appreciated for his valuable supervision, suggestions, supporting, and guidance throughout the course of my study.

This study was supported by the Internet Data Center Operations Department, CAT Telecom Public Company Limited, Thailand.

Finally, I deeply thank my wife, Ms.Panumart Thamthawornsakul who have strongly supported me throughout the period of this research.

Parinya Thamthawornsakul

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Cont.)

LIST OF FIGURES (Cont.)

CHAPTER 1

INTRODUCTION

1.1 STATEMENT AND SIGNIFICANCE OF THE PROBLEMS

Nowadays, Internet is becoming a part of everyday life. Many of communication devices are connected on the Internet. In data communications, Internet provides many benefits to the users. At the same time, there are some users attempt to do damage in various ways on the Internet. In view of security, network access control is required. Most commonly used network access control method is user authentication, which is a method to verify an identity of the users before allowing network access to client devices (authorized device owners). Network access control can be seen as one of functions performing on network equipment. A captive portal [1] and a port-based network access control (IEEE 802.1x) [2] are functions that are widely used in network access control application. Captive portal (also known as hotspot service) focuses on public access control, while port-based network access control focuses on access control in an enterprise. Both access control features have user authentication mechanisms on a basis of session-based authentication.

Session-based authentication is a method to verify user identity before allowing network access based on general data transmission. The user authentication and access control processes are independent. After passing the user authentication process, the access control process remembers source addresses of authorized devices and creates a session that allow only packets sent from those devices to access the network. That means packets sent from client devices do not need to be verified about the identity of the user, until the session expires or is terminated.

Regarding data encapsulation at network layer and lower layers, standard protocol header is not designed to support any security. For example, the protocol header does not have the ability to authenticate user, does not have the ability to check an integrity of the whole packet or frame, does not have the ability to confirm that the data is sent from the real origin, and so on. Therefore, controlling network access with session-based authentication is not able to filter and prevent packets or

frames that are generated by using a technique of source address spoofing [3]. By using such technique, malicious users are able to damage services or devices on the networks by impersonating the identity of some authorized users. It is difficult to find the responsible persons and it is a major problem without any easy solution.

The aforementioned problem brings about studying and applying a sessionless authentication network access control that perform on a network layer. Unlike the session-based authentication, the user identity is attached to the header of every packets sent by device of authorized user. Those packets must be sent to the network access control device to verify the user identity and gain access to the controlled networks. Besides attaching the user identity, additional information is also attached to packet header to make the packet can be verified about data integrity and data origin authentication by itself. These features enhance security in network access control application and provide high accuracy on user identification.

1.2 GOAL AND OBJECTIVE

Sessionless authentication network access control will enable built-in user authentication feature on a packet transmission. User identification will be accurate and reliable. Source-spoofed packets will be detected and denied by network access control device as well as packets that are modified during the transmission. In addition, it will be able to mitigate a replay attack.

1.3 HYPOTHESIS TO BE TESTED

Sessionless authentication can be done by using an HMAC. In term of network security, HMAC provides abilities of data origin authentication and data integrity validation. In packet communication, the HMAC and additional information can be attached to the packet header in part of options. In network attack, a replay attack is able to occur, the Unix time can be used to mitigate the replay attack.

1.4 SCOPE OR LIMITATION OF THE STUDY

The study focuses on how to implement the sessionless authentication network access control for IPv4 communication, in order to prove that each IPv4 packet can be authenticated by itself, packet authentication data can be put into the packet header

at client devices, client device users can be accurately identified at network access control device, and source-spoofed packets or packets with modified data are not allowed access to the controlled networks.

The study is a prove of concept of sessionless authentication network access control. The experiment will point out how to manipulate packets, how to identify users, and how to prevent unwanted packets. Some performance experiments such as data processing performance and network throughput are not in the scope of the study.

1.5 PROCESS OF THE STUDY

The study starts with finding out a possibility to manipulate IPv4 packets flowing in the network stack of the operating system. Then, evaluate a free space of IPv4 options field and consider about which options should be defined and attached to the field, what kind of HMAC can be used. Next, define procedures of manipulating the IPv4 packet header and packet verification. After that, create relevant programs for testing about sessionless authentication, between client device and network access control device, within the experimental environment. Finally, collect and summarize the results.

1.6 ASSUMPTION

The experiment of sessionless authentication network access control is in the server virtualization environment. Source codes of programs used in the experiment do not have any optimization and are not ready for production.

CHAPTER 2

LITERATURE REVIEW

This chapter discusses about related works that will be implemented in the research methodology. Main topics include: sessionless authentication, keyed-hashing for message authentication, Unix time, Internet protocol version 4 and its options, packet manipulation program, and Linux Netfilter framework.

2.1 Sessionless Authentication

Authentication is the process of recognizing an identity of user. It is the mechanism of associating a request with a set of identifying credentials. The credentials provided are compared to those in a database of the authorized user's information within an authentication system. One familiar use of authentication is network access control. A network that is supposed to be accessed only by those authorized must attempt to detect and exclude the unauthorized, this can be done by placing access control devices across the communication path. Access to network is therefore usually controlled by confirming on an authentication procedure to establish confidence the identity of the user, granting access established for that identity.

There are two most common approaches to authentication mechanisms, one of them called session-based (stateful) authentication and the other one is sessionless (stateless or token-based) [4] authentication. When considering network access control, session-based authentication allows packets sent from authorized client devices to access the controlled network, in this case client users have to pass authentication first. While sessionless authentication filters packets attaching user authentication data and allows access only packets having valid user authentication data to access the controlled network. The difference between two authentication mechanisms can be illustrated in figure 2.1.

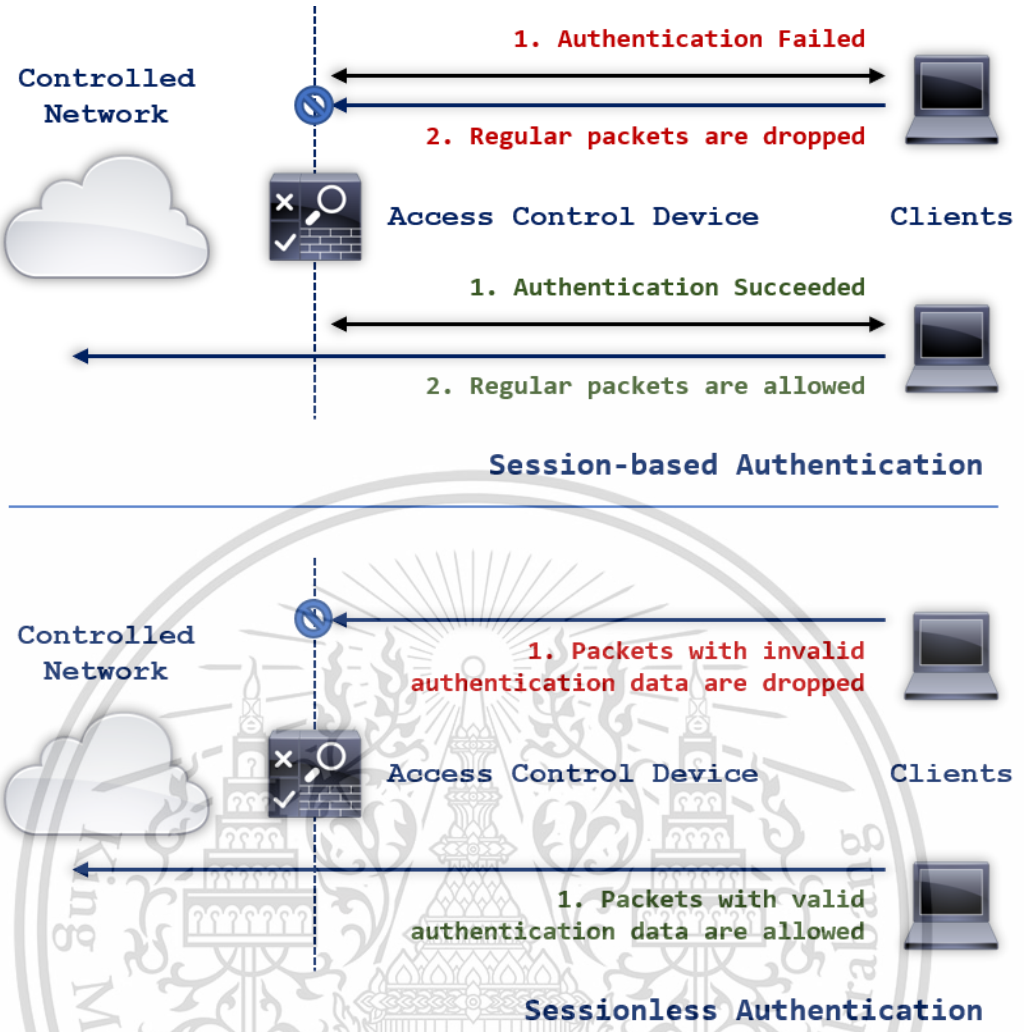


Figure 2.1 Difference between session-based and sessionless authentication

In term of security, sessionless authentication in network access control application is considered it has more secure than session-based authentication. Because all packets that are sent through the access control device must be attached with user authentication data. The user authentication data must be valid and must be verified in a manner of packet per packet. In addition, sessionless authentication also provides a data origin authentication and a data integrity check. These features make possibility to identify client users more accurately and prevent changing information during communication. There are a number of ways to generate and verify user authentication data for sessionless authentication mechanism, one of those ways is using HMAC (Keyed-Hashing for Message Authentication) which will be discussed in the following order.

2.2 HMAC: Keyed-Hashing for Message Authentication

In a field of information security, an HMAC [5] is a specific type of MAC (Message Authentication Code) that provides a data integrity check and a data origin authentication. Its mechanism can be used with any cryptographic hash functions in combination with a secret key. About the cryptographic hash function, data is hashed by iterating a basic compression function on blocks of data. HMAC uses a secret key for computation and verification of the MAC. In computer network, HMAC is used between two communication points (such as between client and server) that share a secret key in order to validate information transmitted between them.

Cryptographic hash function is a mathematical algorithm that maps data of any size to a fixed size and it is designed to be a one-way function which is not possible to invert. MD5 [6] and SHA-1 [7] are examples of cryptographic hash functions. An input data to be computed by the cryptographic hash function is often called the message, and an output is often called the message digest or the hash value. Normally, it should be difficult to find same hash value from different message even a slight change, however this situation is possible to occur which is called a cryptographic hash collision. HMAC reduces the opportunity of such collision. Figure 2.2 illustrates how does cryptographic hash function works.

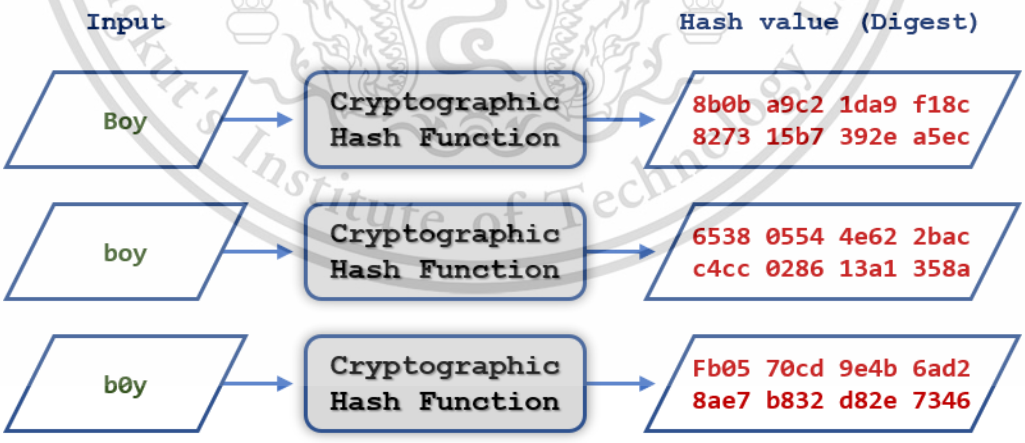


Figure 2.2 How does cryptographic hash function works [8]

HMAC definition requires a cryptographic hash function $\boldsymbol{H}$ and a secret key $\boldsymbol{K}$. Let $\boldsymbol{M}$ denotes a stream of data (message), $\oplus$ denotes an XOR (bitwise exclusive OR

operator), and || denotes a concatenation. To compute HMAC over the message, the HMAC equation is expressed as follows.

$$HMAC(K, M) = H((K \oplus opad) \,||\, H((K \oplus ipad) \,||\, M)) \quad (2.1)$$

In equation 2.1, an ***opad*** and an ***ipad*** are fixed and different hexadecimal constant, ***opad*** is the outer padding value 0x5C repeated 64 times (64-byte string), ***ipad*** is the inner padding value 0x36 repeated 64 times. The secret key can be of any length up to ***opad/ipad*** byte string. The HMAC computation steps are as follows.

1. Append zeros to the end of ***K*** to create a 64-byte string (for example, if ***K*** is of length 20 bytes, then ***K*** will be appended with 0x00 repeated 44 times).
2. XOR the 64-byte string computed in step (1) with ***ipad***.
3. Append the stream of data to the 64-byte string resulting from step (2).
4. Apply ***H*** to the stream generated in step (3).
5. XOR the 64-byte string computed in step (1) with ***opad***.
6. Append the stream computed in step (4) to the 64-byte string resulting from step (5).
7. Apply ***H*** to the stream generated in step (6) and output the result.

The result of HMAC computation will be denoted by HMAC-MD5, HMAC-SHA1, etc. For clarification purposes, the HMAC computation steps can also be illustrated in figure 2.3.

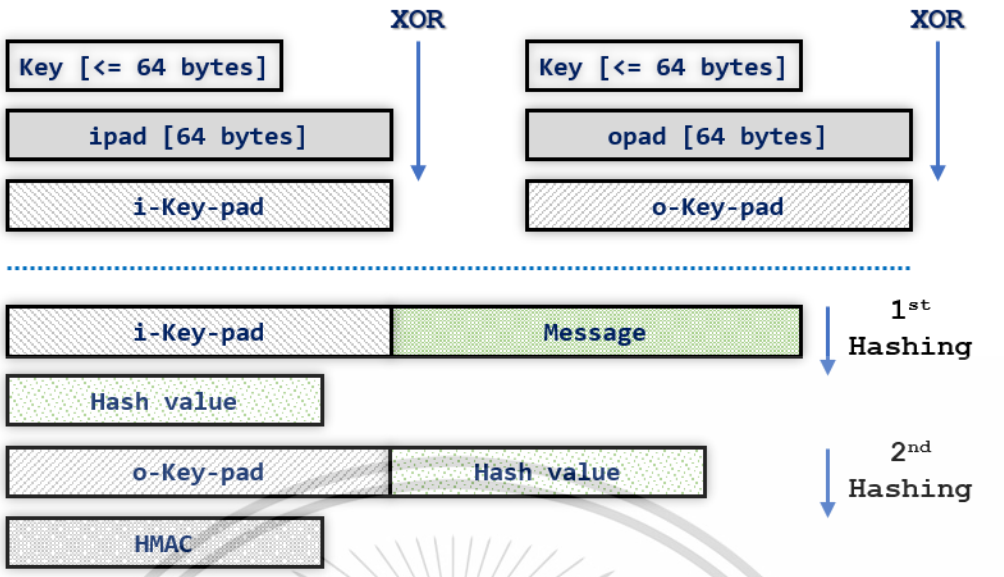


Figure 2.3 Steps of HMAC computation [9]

Regarding network access control, HMAC is a key component for sessionless authentication method, where the integrity of a packet is guaranteed through a code representing the whole packet. However, HMAC is susceptible to replay attack. To mitigate the replay attack, HMAC should be used with a Unix time, which will discuss in the next topic.

2.3 Unix Time and Replay Attack Mitigation

A Unix time (also known as Unix timestamp or POSIX time) is a time measurement system for describing a point in time, defined as the number of seconds that have elapsed since 00:00:00, Thursday 1 January 1970 [10] at GMT (Greenwich Mean Time) which is now referred to as UTC (Universal Time Coordinated or Coordinated Universal Time). Each day is treated as if it contains exactly 86,400 seconds. In computer systems, Unix time is widely used in many operating systems and file formats, datetime is represented as a signed 32-bit number. For example, Sunday 1 July 2018 10:25:59 UTC is represented by number of 1530440759 (1,530,440,759 seconds). The representation will end after the completion of 2,147,483,647 (2^{31} - 1) seconds, because the signed 32-bit number will overflow and will take the actual count to negative number, which will happen on Tuesday 19 January 2038 UTC. However, there are some workable

solutions that will extend the Unix time range, such as changing the data-type from a signed 32-bit number to an unsigned 32-bit number on legacy computer systems and using a signed 64-bit number that is available for computer systems designed to run on 64-bit hardware.

Besides the time measurement system, the Unix time can also be used for replay attack mitigation purpose [11]. Replay attack (also known as playback attack) [12] is a category of network attack. In data communications on computer networks, a valid data transmission is maliciously delayed or repeated by an attacker who intercepts the data and re-transmits it. This is one of versions of a MITM (Man-In-The-Middle) attack. The pattern of replay attack is shown in figure 2.4.

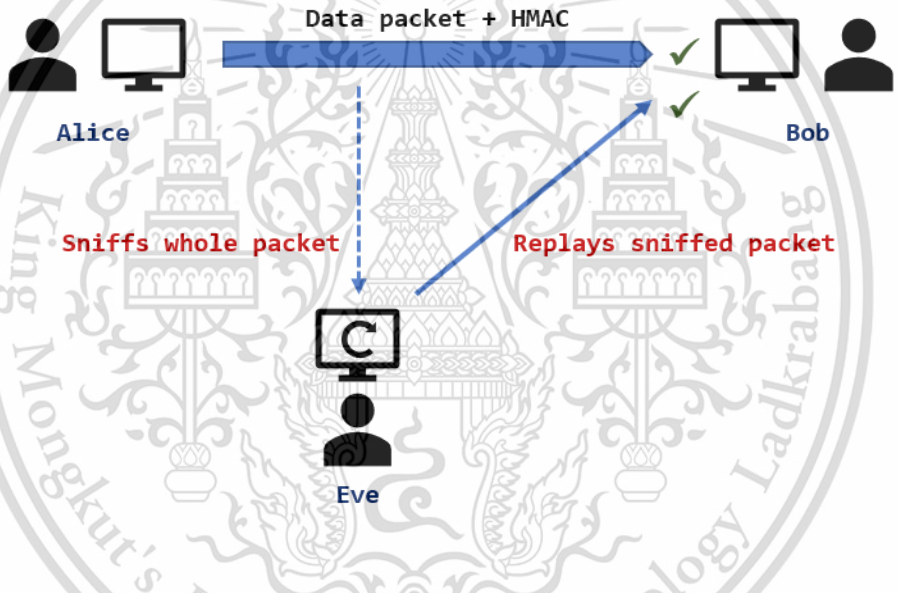


Figure 2.4 Pattern of replay attack

In the figure, Alice wants to prove her identity to Bob for gaining access. Bob requires Alice's credential as a proof of identity, which Alice dutifully provides (after HMAC computation). Meanwhile, Eve is eavesdropping on the communication (between Alice and Bob) and keeps the credential (MAC). After the communication is over, Eve communicates with Bob and uses Alice's credential for a proof of identity. Thus, Bob grants Eve (posing as Alice) access as well.

In order to mitigate the replay attack when using HMAC, the Unix time is inserted as part of data (message) or secret key before entering the HMAC computation. Considering figure 2.4, in a condition of time-synchronized environment, when Alice

wants to communicate with Bob, she always includes her Unix time and credential (MAC that is computed with her Unix time). Bob uses his Unix time to recompute the credential for proofing Alice's identity and only accepts communications that the Unix time is within a reasonable tolerance. In contrast, if it is out of the reasonable tolerant period, Eve cannot communicate with Bob by using Alice's credential. Figure 2.5 shows a benefit of using HMAC with Unix time.

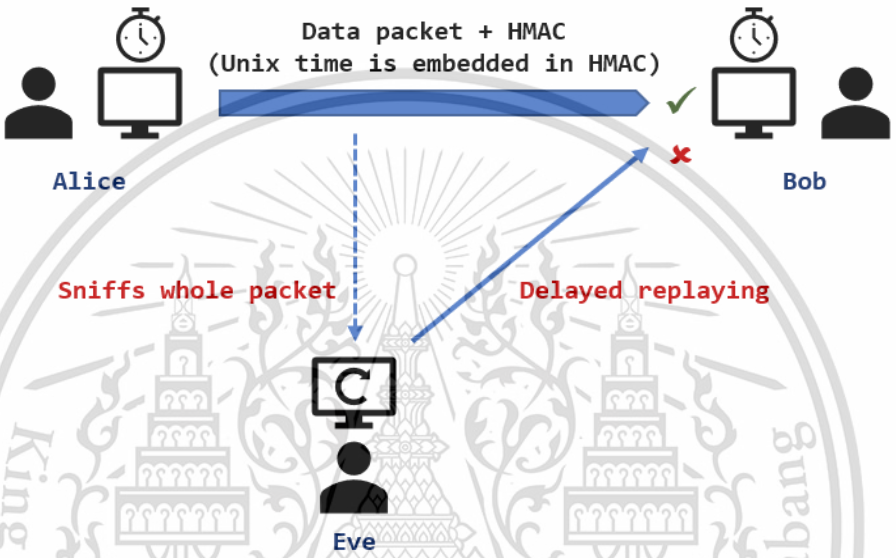


Figure 2.5 Benefit of using HMAC with Unix time

User authentication using HMAC for identifying user credentials can be applied at several levels of data communication. In case of user authentication at network level, the user credentials can be attached to the packet header as options.

2.4 IPv4 and Options

When discussing data communications based on the OSI (Open Systems Interconnection) model [13], the third network-layer protocol is the IP (Internet Protocol). IP is the principal communication protocol for relaying packets across network boundaries. By its routing function, IP enables internetworking and essentially establishes the Internet. IP provides connectionless and unreliable packet delivery because it treats each packet of information independently and it does not guarantee delivery, meaning that it does not require acknowledgments from the sending host,

the receiving host, or intermediate hosts. There are two major versions of IP, the first version, IPv4 (Internet Protocol version 4) [14] is the dominant protocol of the Internet and the successor version, IPv6 (Internet Protocol version 6) [15]. In the next section will discuss the IPv4 and will focus on its options.

IPv4 is one of the core protocols of standards-based internetworking methods in the Internet. It has the task of delivering packet from the source host to the destination host based on the IPv4 address in the packet header. For this purpose, IPv4 defines packet structure that encapsulate the data to be delivered and defines addressing method that are used to label the packet header with source and destination information. IPv4 uses 32-bit addresses which may be represented in any notation expressing a 32-bit integer value. Most often written IPv4 address is in the dot-decimal notation, which consists of four octets of the address expressed individually in decimal numbers and separated by periods (dot). For example, the quad-dotted IP address 192.168.1.254 represents the 32-bit decimal number 3232236030 which in hexadecimal format is 0xC0A801FE, this may also be expressed in dotted hex format as 0xC0.0xA8.0x01.0xFE. IPv4 packet consists of a header section and a data section without data checksum or any other footer after the data section. The IPv4 packet header consists of 14 fields, the first 13 fields are required before sending the IPv4 packet, the fourteenth field is optional which also known as IPv4 options. The header format of IPv4 packet is shown in figure 2.6.

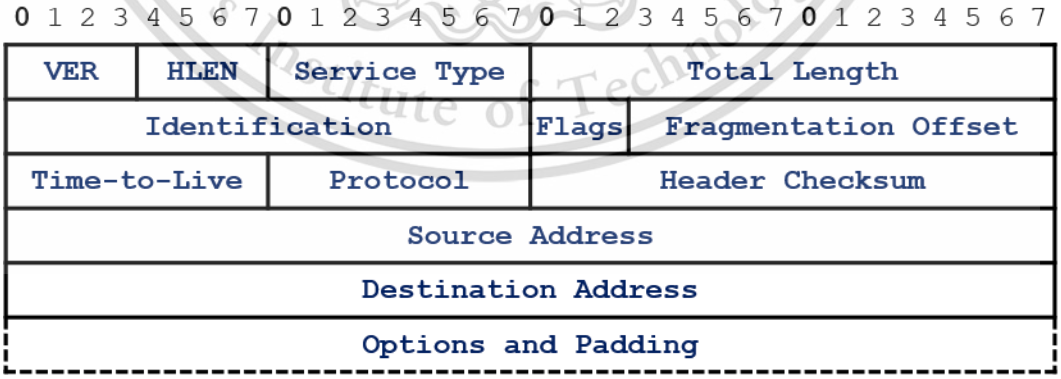


Figure 2.6 IPv4 header format [16]

The packet header is 20 to 60 bytes in length and contains information essential to routing and delivery. A brief description of each field is in order.

1. Version Number (VER). The 4-bit version number field defines the version of the IPv4 protocol, which has the value of 4.
2. Header Length (HLEN). The 4-bit header length field defines the total length of the packet header in 4-byte words. The IPv4 packet has a variable-length header. The receiving host needs to know when the header stops and the data, which is encapsulated in the IPv4 packet.
3. Service Type. In the original design of the IPv4 header, this field was referred to as ToS (Type of Service), afterward IETF (Internet Engineering Task Force) redefined the field to provide DiffServ (Differentiated Services), which defined how the datagram should be handled.
4. Total Length. The 16-bit field defines the total length (IPv4 header plus data) of the IPv4 packet in bytes. This field helps the receiving host to know when the IPv4 packet has completely arrived.
5. Identification, Flags, and Fragmentation Offset. These three fields are related to the fragmentation and reassembly of the IPv4 packet when the size of the IPv4 packet is larger than the underlying network can carry.
6. Time-to-Live (TTL). Due to some malfunctioning of routing protocols an IPv4 packet may be circulating in the Internet, visiting some networks over and over without reaching the destination. The time-to-live field is used to control the maximum number of hops (routers) visited by the IPv4 packet.
7. Protocol. In the data section of an IPv4 packet, called the payload, carries the whole data from upper-layer protocol. The Internet authority has given any protocol that uses the service of IPv4 a unique 8-bit number which is inserted in the protocol field. When the payload is encapsulated in an IPv4 packet, the corresponding protocol number is inserted in this field and when the IPv4 packet arrives at the destination, the value of this field helps to define to which protocol the payload should be delivered.
8. Header Checksum. Due to IPv4 is not a reliable protocol, errors in the IPv4 header can be a disaster. For example, if the destination IP address is

corrupted, the IPv4 packet can be delivered to the wrong host. Therefore, IPv4 adds a header checksum field to check the header.

9. Source and Destination Addresses. These 32-bit source and destination address fields define the IPv4 address of the source host and destination host respectively.
10. Options. An IPv4 header can have up to 40 bytes of options. IPv4 options can be used for network testing and debugging. Although IPv4 options are not a required part of the IPv4 header, option processing is required of the IP software. This means that all implementations must be able to handle options if they are present in the IPv4 header.

The IPv4 header is made of two parts which are a fixed part and a variable part. The fixed part is 20 bytes long and the variable part comprises the IPv4 options that can be a maximum of 40 bytes (in multiples of 4-byte words) to preserve the boundary of the IPv4 header. IPv4 options are divided into two broad categories: single-byte options and multiple-byte options. The single-byte options contain a single octet of option-type, while the multiple-byte options consist of an option-type octet, an option-length octet, and the actual option-data octets. The IPv4 options field formats can be illustrated in figure 2.7.

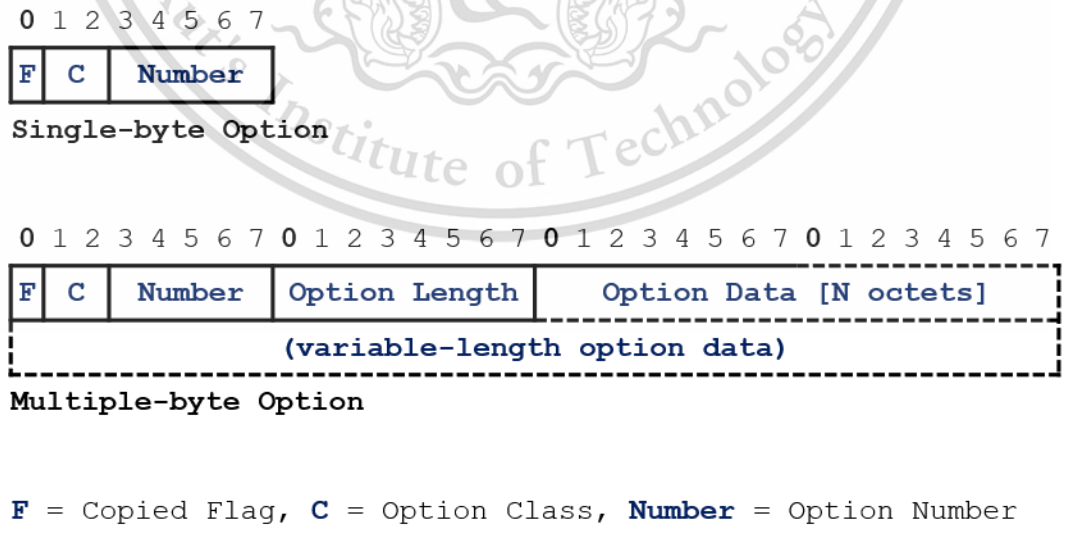


Figure 2.7 IPv4 options field formats

In part of IPv4 options, the option-type octet is viewed as having 3 fields, first 1 bit contains a copied flag, next 2 bits contain an option class, and last 5 bits contain an option number. The copied flag indicates that this option is copied into all fragments on fragmentation, value of 0 means that this option is not copied, and value of 1 means that this option is copied. The option class can be divided into four classes, 0 (00_2) for control, 1 (01_2) is reserved for future use, 2 (10_2) for debugging and measurement, and 3 (11_2) is reserved for future use. The option number can be a maximum value of 31 (11111_2) in each option class. The option-length octet indicates the size of the entire option which counts the option-type octet and the option-length octet as well as the option-data octets. The option-data octets contain option-specific data. Table 2.1 shows IPv4 options [17] specified by IANA (Internet Assigned Numbers Authority).

Table 2.1 IPv4 options specified by IANA

Copy	Class	Number	Name	Description
0	0	0	EOL	End of Option List
0	0	1	NOP	No Operation
1	0	2	SEC	Security
1	0	3	LSR	Loose Source Route
0	2	4	TS	Time Stamp
1	0	5	E-SEC	Extended Security
1	0	6	CIPSO	Commercial Security
0	0	7	RR	Record Route
1	0	8	SID	Stream ID
1	0	9	SSR	Strict Source Route
0	0	10	ZSU	Experimental Measurement
0	0	11	MTUP	MTU Probe
0	0	12	MTUR	MTU Reply
1	2	13	FINN	Experimental Flow Control
1	0	14	VISA	Experimental Access Control
0	0	15	ENCODE	-
1	0	16	IMITD	IMI Traffic Descriptor

Table 2.1 (Cont.)

1	0	17	EIP	Extended Internet Protocol
0	2	18	TR	Traceroute
1	0	19	ADDEXT	Address Extension
1	0	20	RTRALT	Router Alert
1	0	21	SDB	Selective Directed Broadcast
1	0	22	-	Unassigned (Released 18 October 2005)
1	0	23	DPS	Dynamic Packet State
1	0	24	UMP	Upstream Multicast Pkt.
0	0	25	QS	Quick-Start
0	0	30	EXP	RFC3692-style Experiment
0	2	30	EXP	RFC3692-style Experiment
1	0	30	EXP	RFC3692-style Experiment
1	2	30	EXP	RFC3692-style Experiment

There are some unspecified option numbers that can be used for experiment. In order to use such option numbers in packet transmission, a packet header must be modified before sending the whole packet to networks. Due to it is beyond the scope of standard function of the operating system, the third-party program is required, called packet manipulation program.

2.5 Scapy: Packet Manipulation Program

Generally, data communication between hosts based on TCP/IP (Transmission Control Protocol/Internet Protocol) model, network operations on layers lower than the application layer (programs that work with computer networks) are taken by the operating system in part of network stack. On each layer, data is encapsulated, protocol header is labelled by using standard default values. In order to manipulate the header, for example, inserting additional information into the protocol header, or modifying original information in the protocol header, these demands need additional network manipulation programs. In case of IP, it may be called packet manipulation program. There are several programs for manipulating the packet. One of programs that performs very well on a lot of computer network specific tasks is called Scapy.

Scapy [18] is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send packets on the wire, capture packets, match requests and replies, and much more. Scapy can easily handle most network tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery. Scapy is Python program [19] that enables the user to describe a packet or set of packets as layers that are stacked. Fields of each layer have useful default values that can be overloaded. For IPv4, within Scapy shell, after executing ***ls(IP)*** command, the header information that can be manipulated by Scapy is displayed as follows.

Line	Command and Result
1	>>> **ls(IP)**
2	version : BitField (4 bits) = (4)
3	ihl : BitField (4 bits) = (None)
4	tos : XByteField = (0)
5	len : ShortField = (None)
6	id : ShortField = (1)
7	flags : FlagsField (3 bits) = (0)
8	frag : BitField (13 bits) = (0)
9	ttl : ByteField = (64)
10	proto : ByteEnumField = (0)
11	chksum : XShortField = (None)
12	src : SourceIPField (Emph) = (None)
13	dst : DestIPField (Emph) = (None)
14	options : PacketListField = ([])
15	>>>

As the result, Scapy is able to manipulate all fields of IPv4 header. Within the ***IP*** structure, each field has a data type and a standard default value. In case of manipulating IPv4 options, it can be done with the following example.

Line	Command and Result
1	>>> **ip4hdr=IP()**
2	>>> **ip4hdr.show2()**
3	###[IP]###
4	version= 4L
5	ihl= 5L
6	tos= 0x0
7	len= 20
8	id= 1
9	flags=
10	frag= 0L

```
  ttl= 64
  proto= hopopt
  chksum= 0x7ce7
  src= 127.0.0.1
  dst= 127.0.0.1
  \options\
>>> ip4hdr.options=IPOption('%s%s%s'%(
    '\x7a\x06\x01\x02\x03\x04','\x01','\x00'
    ))
>>> ip4hdr.show2()
###[ IP ]###
  version= 4L
  ihl= 7L
  tos= 0x0
  len= 28
  id= 1
  flags=
  frag= 0L
  ttl= 64
  proto= hopopt
  chksum= 0xfbd2
  src= 127.0.0.1
  dst= 127.0.0.1
  \options\
   |###[ IPOption ]###
   |  copy_flag= 0L
   |  optclass= 3L
   |  option= 26L
   |  length= 6
   |  value= '\x01\x02\x03\x04'
   |###[ IPOption_NOP ]###
   |  copy_flag= 0L
   |  optclass= control
   |  option= nop
   |###[ IPOption_EOL ]###
   |  copy_flag= 0L
   |  optclass= control
   |  option= end_of_list
>>>
```

In the example, ***ip4hdr*** variable is defined to contain fields of IPv4 header. Running an ***ip4hdr.show2()*** command displays the detail of IPv4 header with sensible default values (depends on each system). An ***ip4hdr.options=IPOption()*** command is used to modify data in the options field. In the example code, three options are added into the IPv4 header. The first option, the value of option-type octet is 0x7A (01111010_2), meaning that, this option will not be copied on fragmentation, it is assigned to the reserved class, option number is 26, this option has 6 octets (bytes)

long, and actual data consist of 0x01, 0x02, 0x03, and 0x04. The last two options (a single octet of option-type) are NOP (No Operation, 00000001_2) and EOL (End of Option List, 00000000_2), respectively. When running the ***ip4hdr.show2()*** command again, the detail of modified IPv4 header is displayed. Because the added IPv4 options have total 8 octets long, plus 20 octets of original IPv4 header, therefore the values of ***ihl*** (Header Length) and ***len*** (Total Length) are changed, and ***chksum*** (Header Checksum) is recalculated.

The example described above, packet is generated and modified by Scapy program, but in real work Scapy will be used to modify packets generated by other applications. Hence the packets must be intercepted and redirected from network stack to the packet manipulation program. The next topic will discuss about how to intercept and redirect the flow of packet within operating system.

2.6 Linux Netfilter Framework

In computer security, a packet filtering framework (also known as host-based firewall) is an important part the operating system that monitors and controls incoming and outgoing network traffic of host computers, based on predetermined security rules. In the next order will discuss about packet filtering framework on Linux operating system called Netfilter.

Netfilter is a very important part of the Linux kernel [20] in term of security, packet mangling, and manipulation. The front-end for Netfilter is iptables, which is a program that tells the kernel what the user wants to do with the packets arriving into, passing through, or leaving the host. Most used features of Netfilter are packet filtering and network address translation, but there are a lot of other things that Netfilter can do, such as controlling network traffic flowing between network stack and application within the same host, this feature is needed to work with packet manipulation program (Scapy). Netfilter has a few tables, each table contain a default set of rules, which are called chains. An overview of how packets travel through the tables and their chains is shown in figure 2.8.

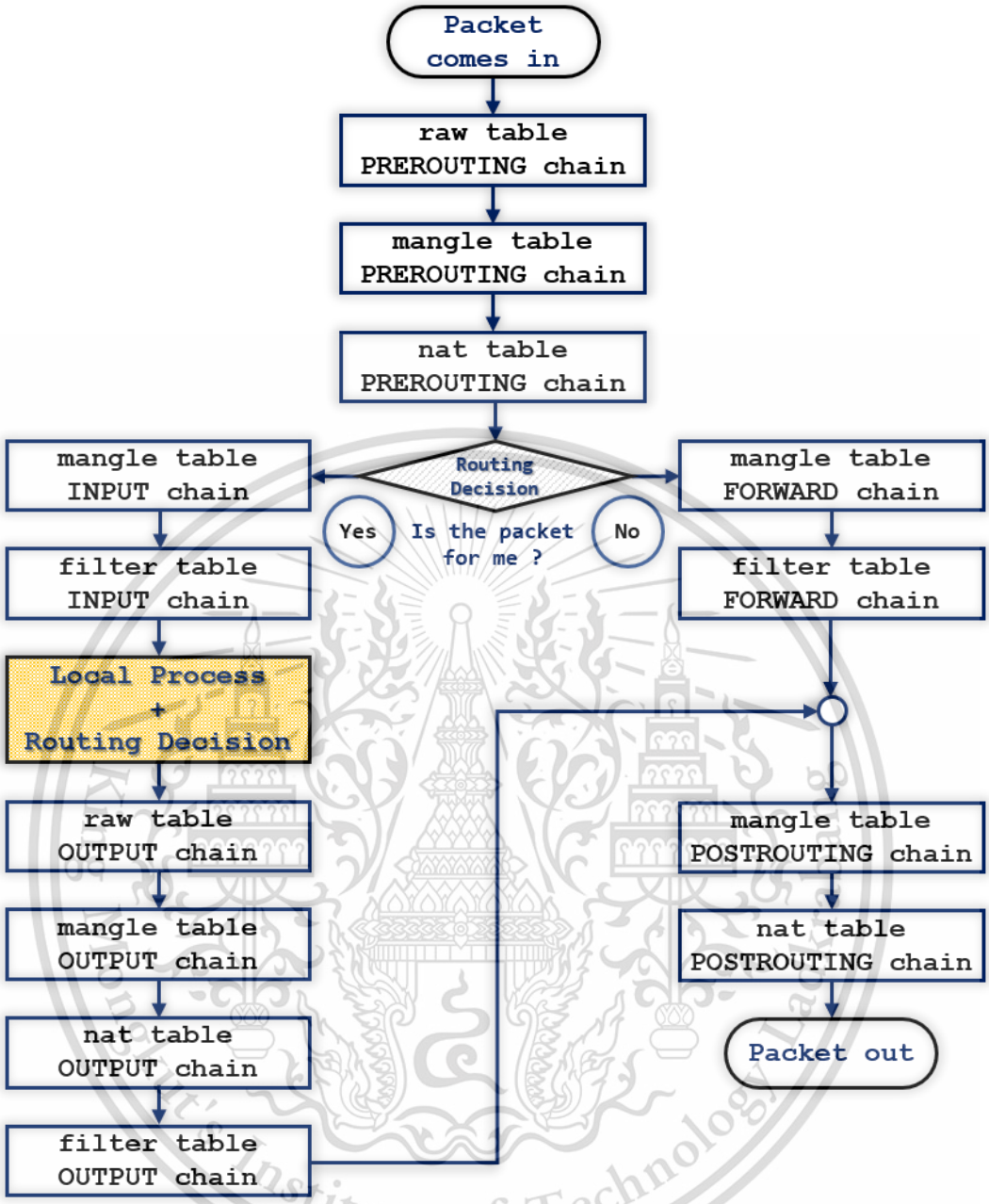


Figure 2.8 Tables and chains in Netfilter framework

As shown in the figure, there are four tables named raw, mangle, nat, and filter. Each table has different functionality. The raw table is used before any other tables in Netfilter. Due to Netfilter can be seen as stateful firewall, which means that packets are inspected with respect to their state. The raw table allows Netfilter to work with packets before the kernel starts tracking state of the packets (can be seen as stateless firewall). The mangle table is used mainly for mangling packets (alter packet headers

in various ways). The nat table is used mainly for NAT (Network Address Translation) purpose, by changing the source and destination addresses of packets. The filter table is used mainly for making decisions about whether a packet should be allowed to reach its destination. In part of chains, each of Netfilter tables are composed of a few default chains. These chains filter packets at various points. Matching rules in the PREROUTING chain apply to packets as they just arrive on the network interface. Matching rules in the INPUT chain apply to packets before they are given to a local process (network-related application). Matching rules in the OUTPUT chain apply to packets after they have been produced by the local process. Matching rules in the FORWARD chain apply to any packets that are routed through the current host. The last chain, POSTROUTING, the matching rules in this chain apply to packets as they just leave the network interface. A brief explanation on how Netfilter works is as follows.

1. The user instructs the kernel about what it needs to do with the packets that flow through the host using the iptables program.
2. The host then analyzes the IP headers on all packets flowing through it.
3. When looking at the IP headers, the kernel finds matching rules (including how to proceed with a packet), if any packets match the matching rules then the packet is manipulated according to the matching rule.

For example, in order to redirect outgoing packets from network stack of the host to packet manipulation program, the following iptables command is needed.

Line	Command
1	`# iptables -t mangle -A POSTROUTING -o ens160 \` `  -j NFQUEUE --queue-num 0`

By using the command above, packets that are being sent to physical network (via network interface named ens160) are intercepted and redirected to the user space program via NFQUEUE number 0. NFQUEUE (also known as Netfilter Queue) is an iptables command target that delegate the decision on packets flowing in a kernel space to a user space program. In part of packet manipulation program, Scapy is used as a library for providing packet manipulation functions. The following source code is

an example of packet manipulation program written in Python that communicates with NFQUEUE and displays packet information.

Line	Source Code
1	`#!/usr/bin/python`
2	`import socket`
3	`import nfqueue`
4	`from scapy.all import *`
5	`def callback_fn(payload):`
6	`    data = payload.get_data()`
7	`    packet = IP(data)`
8	`    packet.show2()`
9	`    payload.set_verdict_modified(nfqueue.NF_ACCEPT, str(packet), len(packet))`
10	`def main():`
11	`    q = nfqueue.queue()`
12	`    q.open()`
13	`    q.bind(socket.AF_INET)`
14	`    q.set_callback(callback_fn)`
15	`    q.create_queue(0)`
16	`    try:`
17	`        q.try_run()`
18	`    except KeyboardInterrupt:`
19	`        q.unbind(socket.AF_INET)`
20	`        q.close()`
21	`main()`

On a host that is running packet manipulation program (with the code above), when a user attempts to send an ICMP (Internet Control Message Protocol) packet out to the destination, the program displays packet information as follows.

Line	Result
1	`###[ IP ]###`
2	`  version   = 4L`
3	`  ihl       = 5L`
4	`  tos       = 0x0`
5	`  len       = 84`
6	`  id        = 41433`
7	`  flags     = DF`
8	`  frag      = 0L`
9	`  ttl       = 64`
10	`  proto     = icmp`
11	`  chksum    = 0x464a`
12	`  src       = 122.155.203.222`
13	`  dst       = 8.8.4.4`

```
  \options\
###[ ICMP ]###
     type      = echo-request
     code      = 0
     chksum    = 0x3db
     id        = 0x584
     seq       = 0x1
```

Linux Netfilter framework and NFQUEUE are suitable for packet customization during transmission and other experiments about networking. The condition is that they must be used with NFQUEUE compatible programs in order to modify packet header, drop packets, accept packets from and return packets to network stack. In case of Scapy, these functions can be done by calling the call-back function.

CHAPTER 3

RESEARCH METHODOLOGY

This chapter discusses about how to implement a sessionless authentication network access control. Firstly, unspecified IPv4 options must be defined to be compliant with the requirements of user authentication, data origin authentication, data integrity verification, and replay attack mitigation. The main equipment used in the experiment is divided into two parts: a client device which is a user device that need access to controlled networks and an authenticator which is a server computer that perform as network access control device. The experiment is in a virtualization environment. The test packets will be captured by a sniffer and will be used as the results of the experiment. The details discuss in the next order.

3.1 IPv4 Option Specifications for Sessionless Authentication

In part of IPv4 options field, new IPv4 options are defined on the basis of using the reserved option classes and the experimental option numbers. Thus, option class 1 and 3, and option number 26 to 29 are suitable for research and experiment. IPv4 option specifications are as follows.

1. User Identification Option. This option is used to identify client device users that need to send packets to restricted networks, pass through the authenticator. Identification information of each user is mapped with a secret key and is stored in a user account database. In action, identification information is the index of the secret key that is used as one of input data for HMAC computation. The user identification option is a multiple-byte option, which is not copied on fragmentation and is set to option class 3 and option number 26, total length of the option is 6 bytes, there are 4 bytes space for containing the user identification data. The structure of user identification option can be illustrated in figure 3.1.

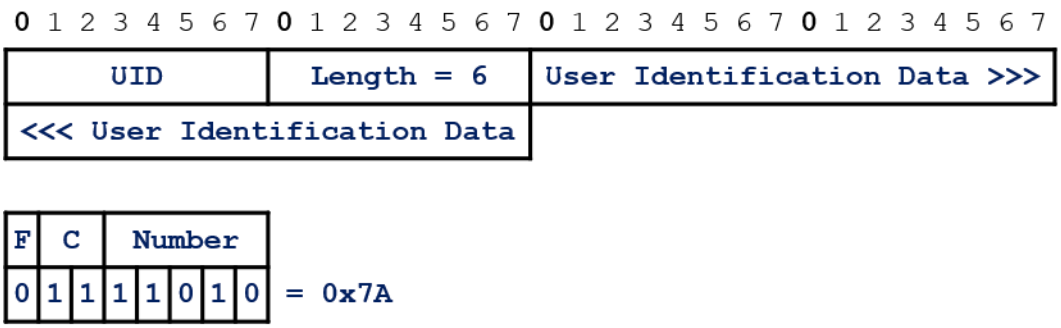


Figure 3.1 User identification option format

2. Timestamp Option. In term of replay attack mitigation, this option contains the Unix time of client devices, it will be compared to the Unix time of the authenticator. Packets received by the authenticator will be processed in next steps only if they have reasonable time value. The Unix time is also part of input data for HMAC computation. The timestamp option is a multiple-byte option, which is not copied on fragmentation and is set to option class 3 and option number 27, total length of the option is 6 bytes, there are 4 bytes space for containing the Unix time value. The structure of timestamp option can be illustrated in figure 3.2.

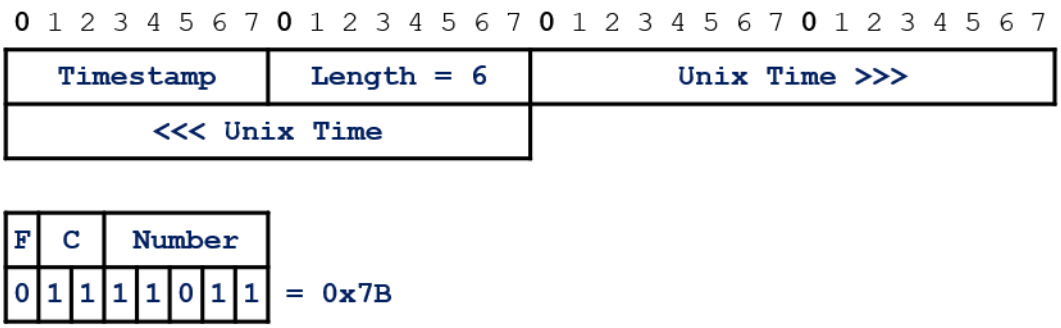
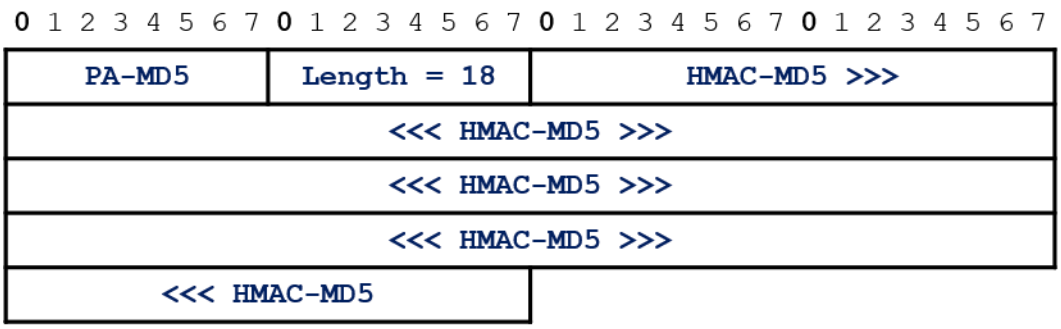
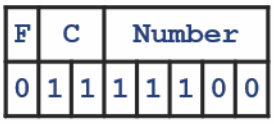


Figure 3.2 Timestamp option format

3. Packet Authentication Option. The purpose of this option is to provide data integrity check and data origin authentication for packets. The packet authentication option is a multiple-byte option, which is not copied on fragmentation and is set to option class 3 and option number 28, total length of the option can be either 18 bytes or 22 bytes, the length of option-data can be either 16 bytes for containing HMAC-MD5 or 20 bytes for containing HMAC-SHA1. HMAC computed by client device will be compared to HMAC recomputed by the authenticator (by using input data from the same packet) for packet authentication. The structure of packet authentication option can be illustrated in figure 3.3 and 3.4.



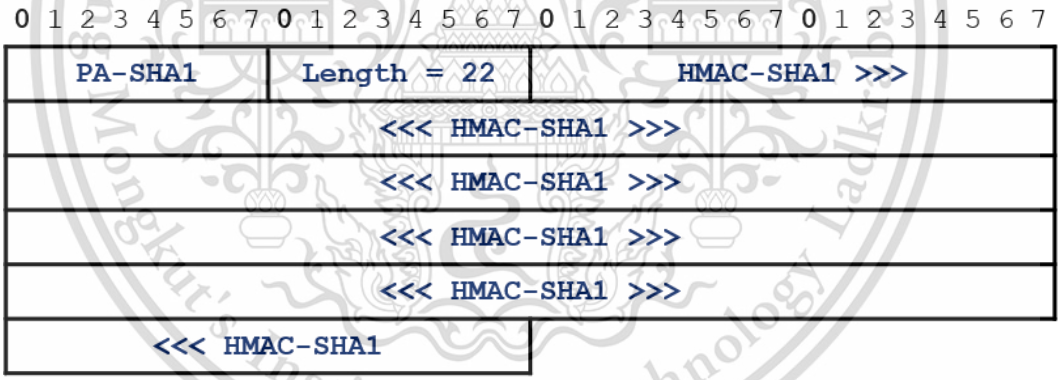


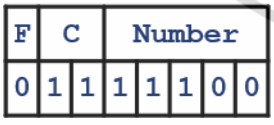
F = 0_2 = The option is not copied on fragmentation

C = 11_2 = Option Class 3

Number = 11100_2 = Option Number 28

Figure 3.3 Packet authentication option format (MD5)





F = 0_2 = The option is not copied on fragmentation

C = 11_2 = Option Class 3

Number = 11100_2 = Option Number 28

Figure 3.4 Packet authentication option format (SHA1)

When combining all new IPv4 options with the fixed part of IPv4 header, the overview of IPv4 packet can be shown in figure 3.5 and 3.6.

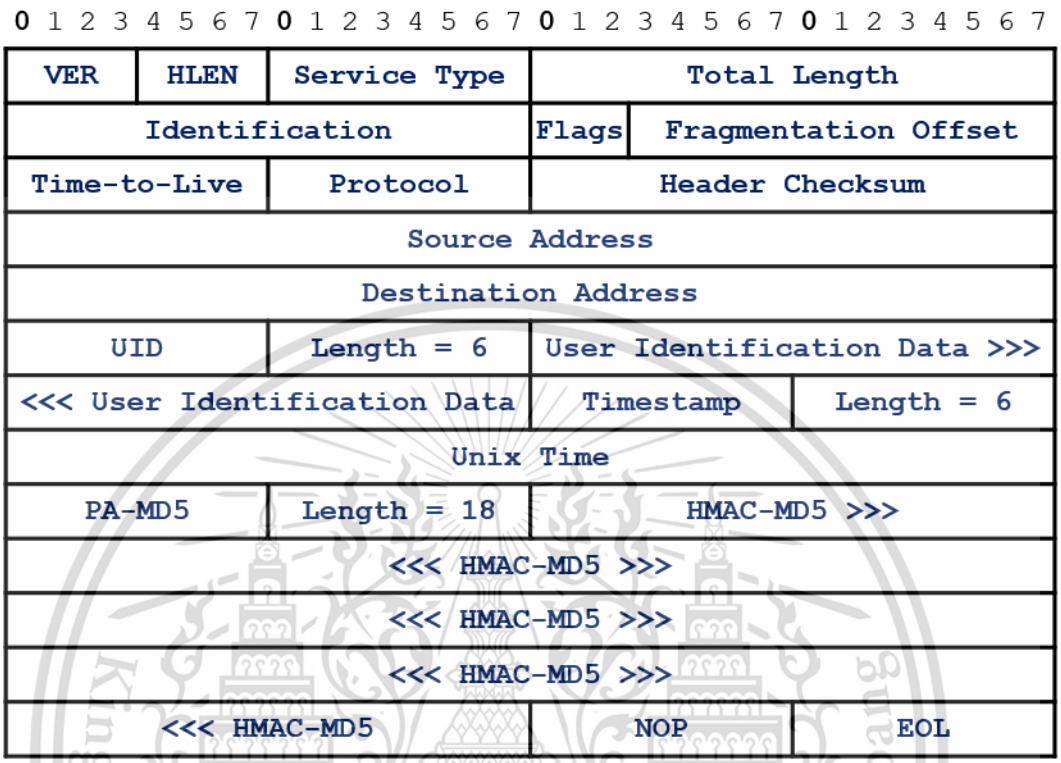


Figure 3.5 IPv4 header with options (MD5)

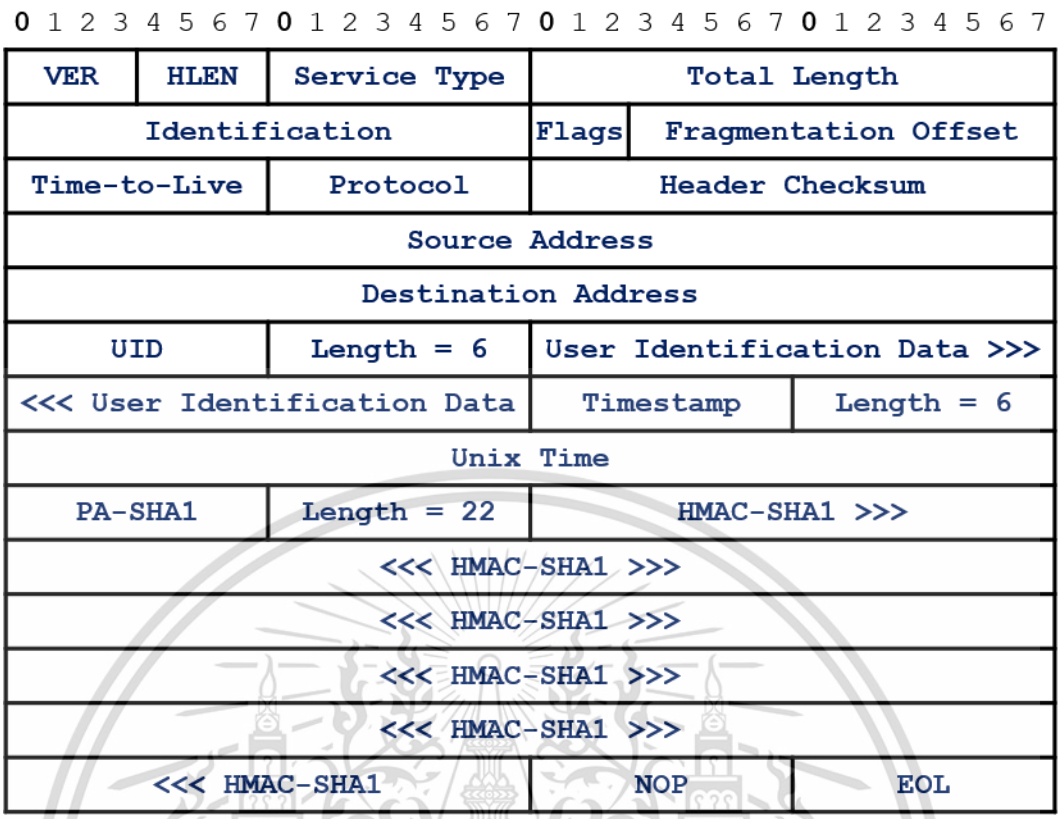


Figure 3.6 IPv4 header with options (SHA1)

In the figure, due to putting IPv4 options must be aligned on a 32-bit boundary, therefore the options field must be padded with NOP and EOL options respectively. In part of HMAC computation, data in immutable fields, a value of Unix time, and IPv4 payload data are used as input data in a form of concatenation. The immutable fields of IPv4 header consist of the version number field, the header length filed, the total length field, the identification field, the protocol field, the source address field, and the destination address field. The secret keys for HMAC computation are referenced by the user identification data. Regarding the header length field, when choosing HMAC-SHA1 for HMAC computation, the length is longer than choosing HMAC-MD5.

3.2 Network Traffic Control

About the Netfilter framework that is mentioned previously, almost all research methods are conducted on Linux operating system. After defining the new IPv4 options for attaching to IPv4 header or removing from IPv4 header. To make it work practically, IPv4 packets that are being forwarded inside the Linux kernel, must be sent (forcing)

to the packet manipulation program. The Netfilter framework is used to accommodate this requirement. In the experiment, there are two main scenarios for controlling the flow of network traffic: controlling the direction of IPv4 packets of client devices before sending to the network and controlling the direction of IPv4 packets flowing through the authenticator before sending to packet's destinations.

At client device, IPv4 packets are generated by local process. Then the packets are forwarded through chains and tables of the Netfilter framework as follows: OUTPUT chain of raw table, OUTPUT chain of mangle table, OUTPUT chain of nat table, OUTPUT chain of filter table, POSTROUTING chain of mangle table, and POSTROUTING chain of nat table. After that the packets are sent to the physical network. At POSTROUTING chain of mangle table, an outbound traffic is intercepted and redirected to the packet manipulation program via NFQUEUE for attaching IPv4 options to outgoing IPv4 packets, then the modified packets are sent back to such chain and table via the same NFQUEUE and are forwarded along the normal flow. Network traffic control at client device can be illustrated in figure 3.7.

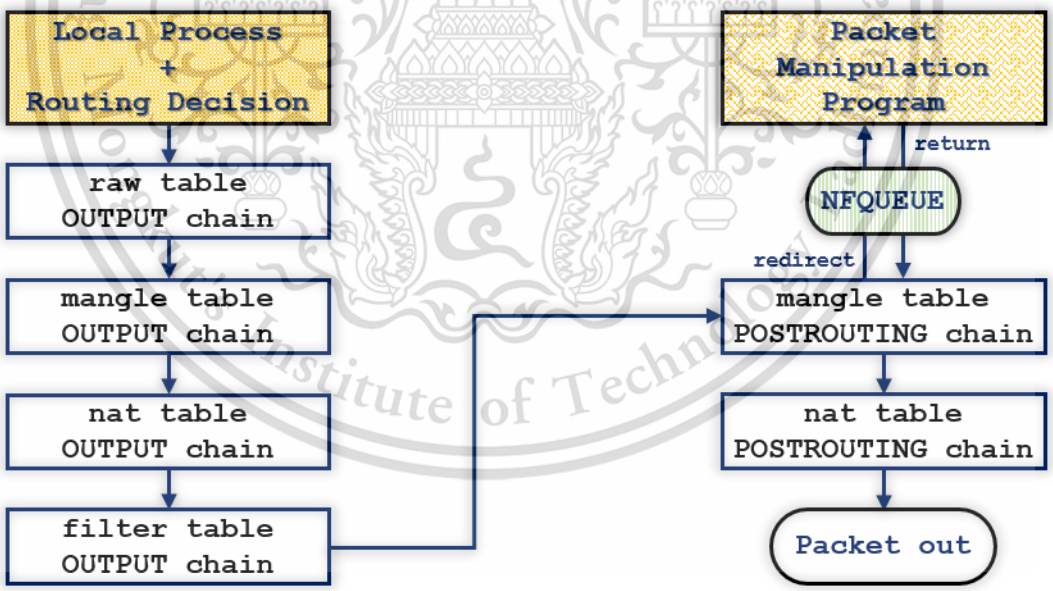


Figure 3.7 Network traffic control at client device

At authenticator, IPv4 packets can be forwarded through itself in two modes: route mode (the packets are routed to other networks) and bridge mode (the packets are bridged to the same network across the authenticator). However, in Netfilter

framework, network traffic of all modes is forwarded through chains and tables in the same manner. IPv4 packets received by inbound network interface are forwarded through the following points: PREROUTING chain of raw table, PREROUTING chain of mangle table, PREROUTING chain of nat table, FORWARD chain of mangle table, FORWARD chain of filter table, POSTROUTING chain of mangle table, and POSTROUTING chain of nat table. Finally, the packets are sent to the physical network by outbound network interface. At POSTROUTING chain of mangle table, only outbound traffic is intercepted and redirected to the packet manipulation program via NFQUEUE for verifying IPv4 packets and removing IPv4 options from the valid packets, then the packets (without IPv4 options) are sent back to such chain and table via the same NFQUEUE and are forwarded along the normal flow. Network traffic control at authenticator can be illustrated in figure 3.8.

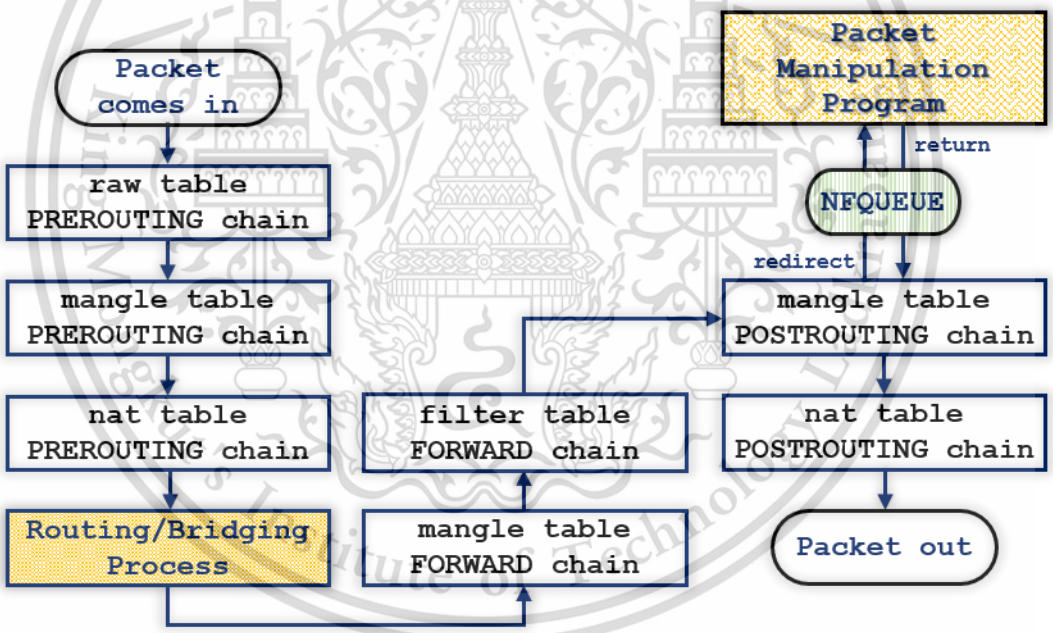


Figure 3.8 Network traffic control at authenticator

3.3 Packet Manipulation and Operations

In part of packet manipulation, redirected IPv4 packets are modified or verified by packet manipulation program which is a program written in Python. The Scapy is imported as a library in the program for providing packet manipulation functions. At client device, packet manipulation program is responsible for attaching IPv4 options to

the packet header. At authenticator, the program is responsible for verifying IPv4 packets sent by client devices, dropping invalid packets, and removing IPv4 options from valid packets. A manipulation of IPv4 packet header at client device can be described as follows.

1. Receive the packet from the NFQUEUE.
2. Change the header length (HLEN) value to the original value plus 8 (32-bit boundary) in case of using HMAC-MD5 for packet authentication data, or plus 9 in case of using HMAC-SHA1.
3. Change the total length value to the original value plus 32 (bytes) in case of using HMAC-MD5 for packet authentication data, or plus 36 in case of using HMAC-SHA1.
4. Compute the HMAC using the secret key mapped with the user identification data and the message consisting of value of header fields and data as follows: version number, header length, total length, identification, protocol, source address, destination address, payload, and Unix time.
5. Attach the user identification option, the timestamp option, and the packet authentication option (HMAC-MD5/HMAC-SHA1) to the IPv4 packet header.
6. Recalculate the header checksum.
7. Send the packet back to the NFQUEUE.

When the IPv4 packets sent from the client devices arrive at authenticator, those packets must enter the packet verification process before forwarding to the packet's destinations. A manipulation of IPv4 packet header at authenticator can be described as follows.

1. Receive the packet from the NFQUEUE.
2. **Drop** the packet if the number of attached options is less than 3 (User Identification, Timestamp, and Packet Authentication).
3. **Drop** the packet if the option number is not 26 or 27 or 28.
4. **Drop** the packet if the value of timestamp option exceeds the acceptable period of the authenticator (±1 second is assigned in the experiment).
5. Find the secret key associated with the user identification data in a dictionary (array variable), search for secret key from an external database in case of it

is not found in the dictionary, keep the secret key in the dictionary if it is found in the external database or **drop** the packet if there is no secret key found.

6. Compute the HMAC using the secret key in the dictionary in the same method as the client device. The value of option length (packet authentication option) is used to select the corresponding HMAC for computation.
7. Compare the computed HMAC to the value of the packet authentication option. **Drop** the packet if both values do not match.
8. Remove all attached options from the packet and change the header length to the value of 5.
9. Recalculate the total length of the packet and the header checksum.
10. Send the packet back to the NFQUEUE.

3.4 Experimental Environment

The experimentation of sessionless authentication network access control is a simulation of packet transmission. Several forms of IPv4 packet are sent from the client device to hosts in the Internet (the controlled networks). An authenticator is placed across the communication path. It is able to work in both bridge mode and routing mode. An operating system running on both client device and authenticator is Linux. The basic specifications are as follows.

- Operating system: *Ubuntu Linux 18.04*
- Network interface card: *2 for authenticator*, *1 for client device*
- Programming language: *Python*
- Additional programs (software package name) required for implementation: *iptables*, *bridge-utils*, *python-scapy*, *python-nfqueue*, *python-sqlite*, and *tcpdump*.

The overall experimental environment (bridge mode authenticator) is shown in figure 3.9.

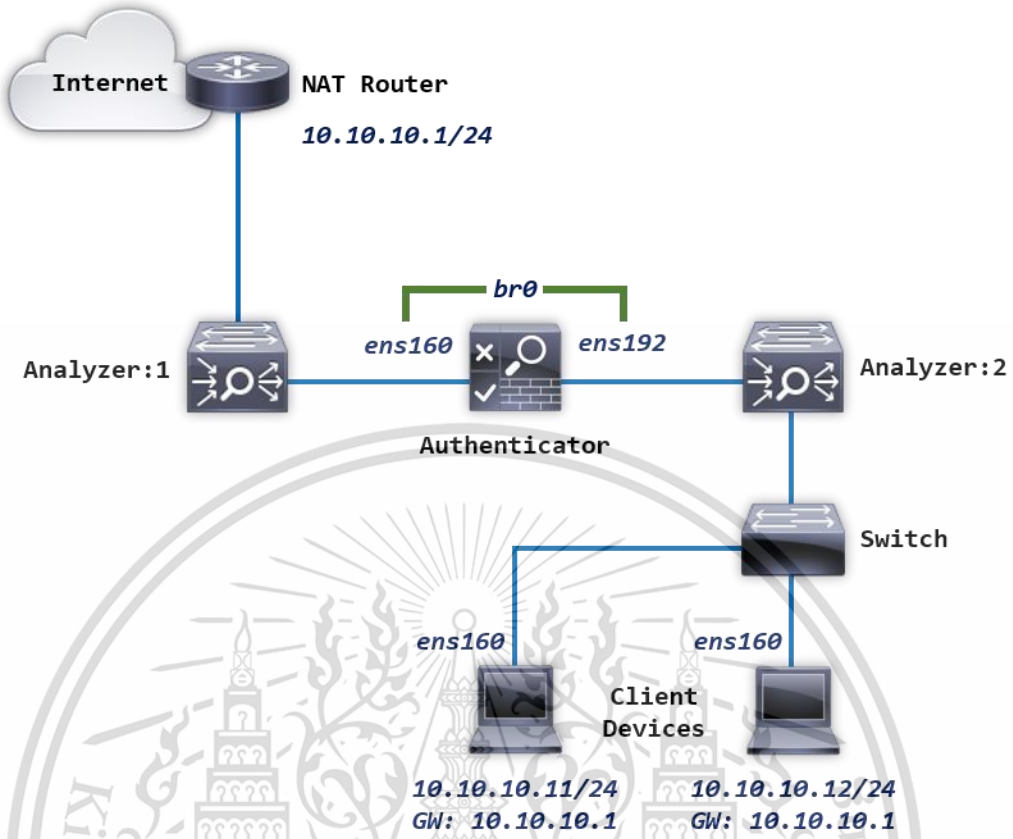


Figure 3.9 Network topology and devices for the experiment (bridge mode)

In bridge mode, network interface cards of the authenticator are bridged, the name of bridge interface is ***br0***. There is only one private network in the network topology. IPv4 packets sent from client devices will be forwarded through the authenticator before they are routed to the Internet. Netfilter ***iptables*** commands used for intercepting and redirecting packets from kernel space to packet manipulation programs on the client device and the authenticator are as follows.

Line	Command (Client Device)
1	`# iptables -t mangle -A POSTROUTING -o ens160 \` `  -j NFQUEUE --queue-num 0`

Line	Command (Authenticator)
1	`# iptables -t mangle -A POSTROUTING -m physdev \` `  --physdev-out ens160 -j NFQUEUE --queue-num 0`

In routing mode, functions of sessionless authentication network access control still same as bridge mode. There are two networks between the authenticator, routing function is an additional task of the authenticator in this mode. For tasks of packet manipulation, ***iptables*** command used on the client device in bridge mode is used on both client device and authenticator in the routing mode. The experimental environment (routing mode authenticator) is shown in figure 3.10.

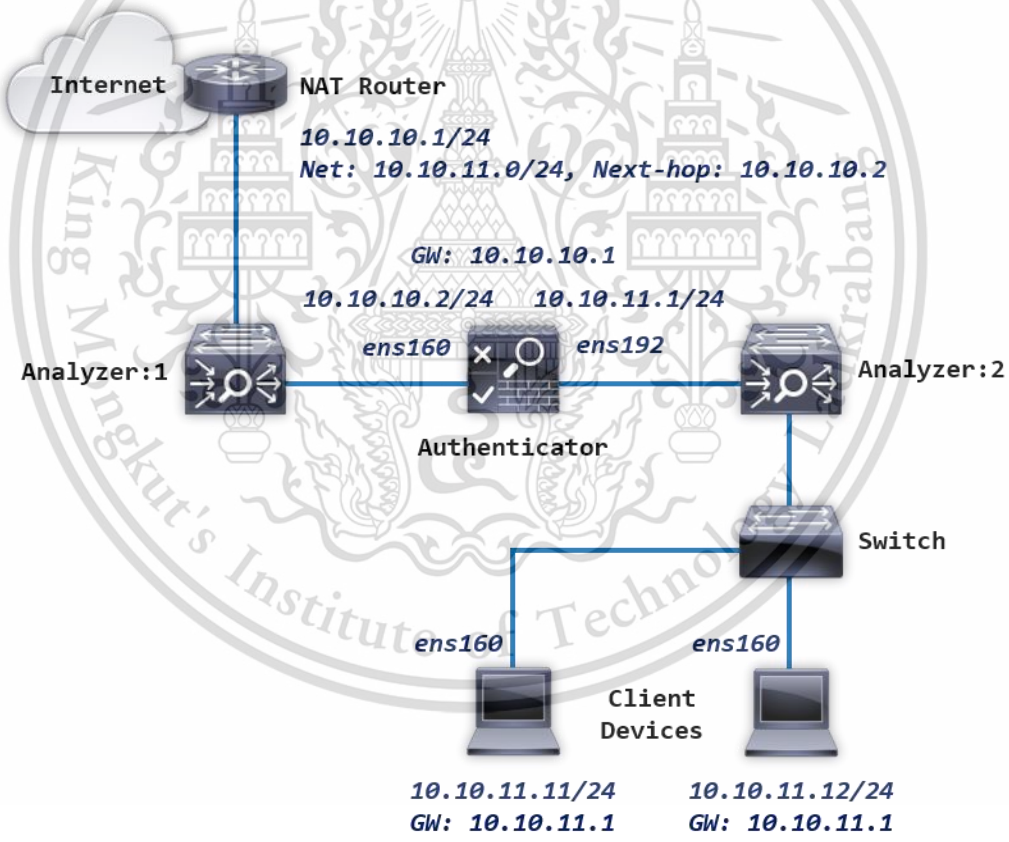


Figure 3.10 Devices and network topology for the experiment (routing mode)

In the experimental environment of both modes, analyzer devices are placed on both sides of the authenticator. Captured packets on the analyzers and packet

manipulation program output displayed on the Linux console are used as an experimental result.

3.5 Experimental Scenarios Simulation

The following are main scenarios which will be simulated to see the action with the IPv4 packets on both client device and authenticator and to test about features of sessionless authentication network access control.

- Test of general packet transmission: at client device, the packet will be sent to the destinations without any IPv4 options.
- Test of IPv4 option numbers validation: at client device, the packet will be sent to the destinations with incorrect option numbers (number 26, 27, and 28).
- Test of timestamp validation: at client device, the packet will be sent to the destinations with a value of timestamp option that exceeds the acceptable value configured on the authenticator.
- Test of secret key management on the authenticator: at client device, the packet will be sent to the destinations with user identification data that is not in the database.
- Test of packet authentication: at client device, both valid packet and invalid packet (using a wrong secret key in HMAC computation process) including options will be sent and forwarded through the authenticator.

CHAPTER 4

EXPERIMENTAL RESULTS AND DISCUSSION

4.1 Experimental Results of Bridge Mode Network Access Control

In the experiment of general packet transmission, IPv4 packet (with identification value ***15238***) sent from the client device without attaching options required for sessionless authentication network access control is dropped at the authenticator. The experimental results are shown in figure 4.1 and 4.2.

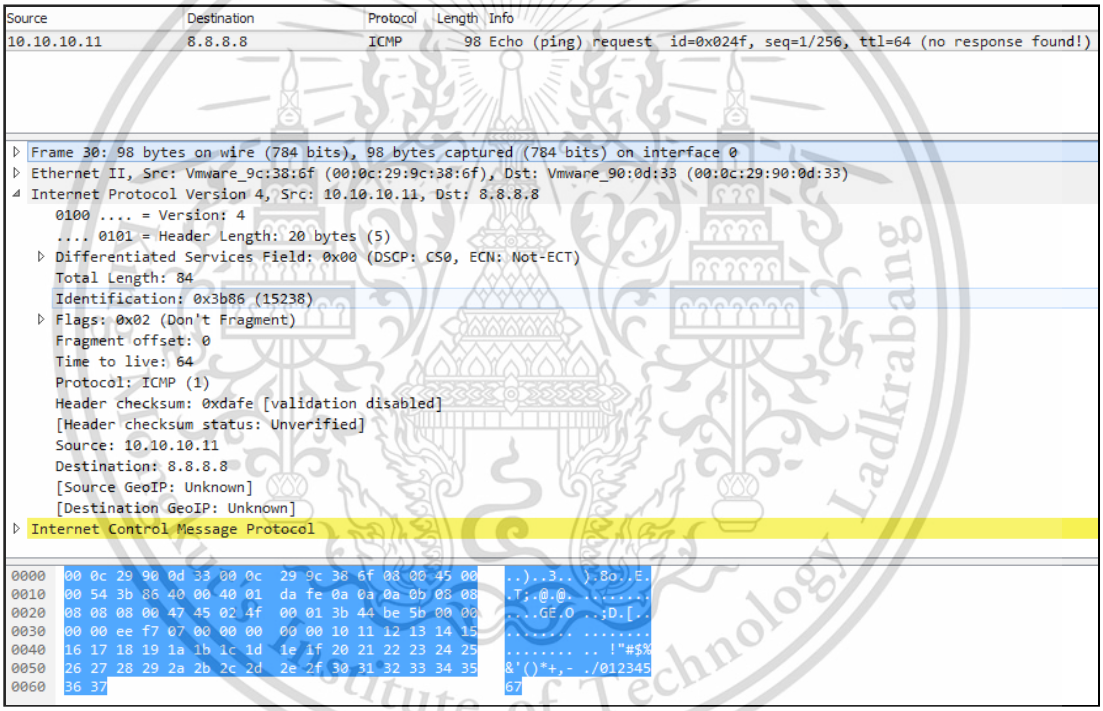


Figure 4.1 The detail of captured packet on the analyzer 2

```
Packet SRC Address: 10.10.10.11
Packet DST Address: 8.8.8.8
Packet HDR Length (before processing): 5 [32-bit words]
Packet ALL Length (before processing): 84 [bytes]
Packet Identification: 15238
Authentication: DROP (the amount of required IPv4 options is not enough)
```

Figure 4.2 Packet manipulation program output displayed at the authenticator

In the experiment of IPv4 option numbers validation, IPv4 packet (with identification value ***45822***) sent from the client device with options: ***0x7A***, ***0x7B***, and ***0x7D*** is dropped at the authenticator because the option ***0x7D*** is not defined to use with sessionless authentication network access control. The experimental results are shown in figure 4.3 to 4.5.

```
Packet SRC Address: 10.10.10.11
Packet DST Address: 8.8.8.8
Packet HDR Length (before processing): 5 [32-bit words]
Packet ALL Length (before processing): 84 [bytes]
Packet Identification: 45822
User ID Number: 56601420 [Integer], 035fab4c [Fixed HEX]
User Unix Time: 1539200073 [Integer], 5bbe5449 [Fixed HEX]
Packet AUTH Data: 83ec14db67f0dacf5065772cdc6a57dcbbb063c2 [HMAC-SHA1]
Packet HDR Length (after processing): 14 [32-bit words]
Packet ALL Length (after processing): 120 [bytes]
```

Figure 4.3 Packet manipulation program output displayed at the client device

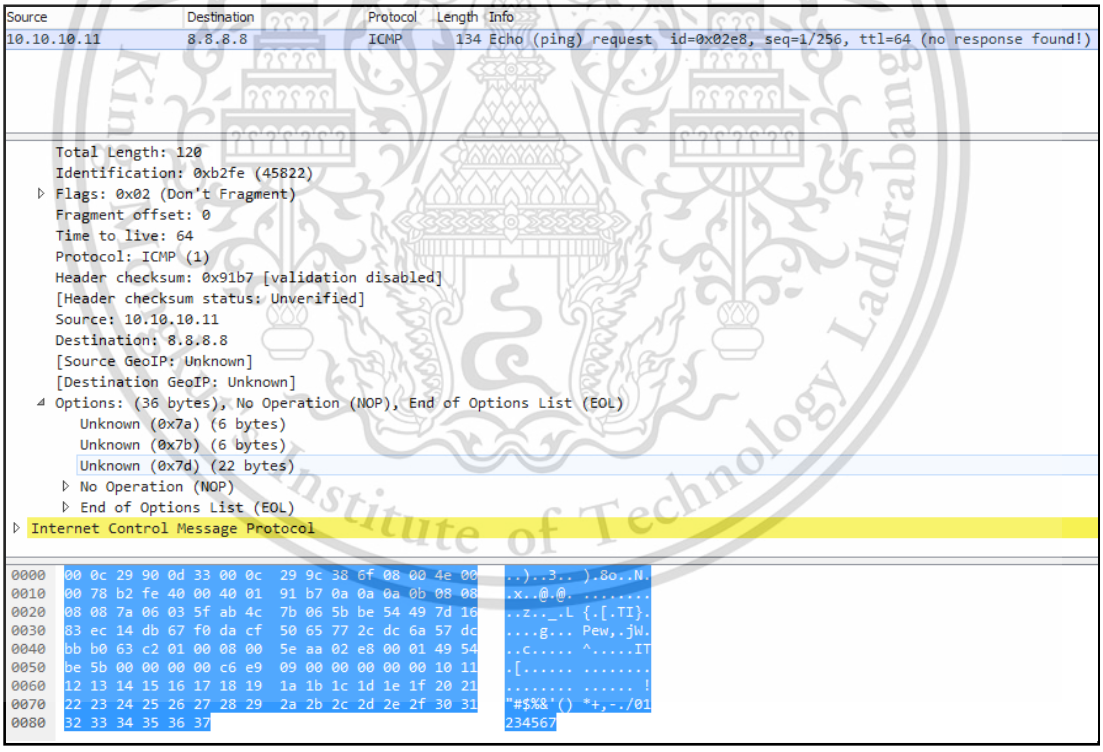


Figure 4.4 The detail of captured packet on the analyzer 2

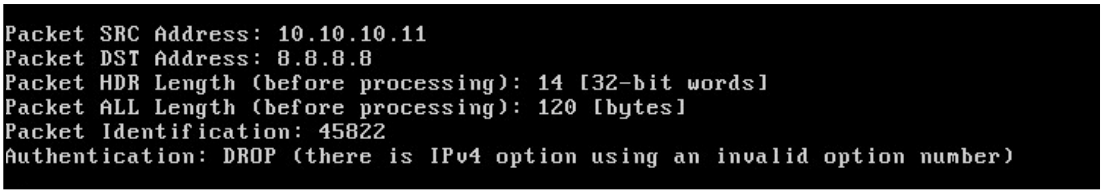


Figure 4.5 Packet manipulation program output displayed at the authenticator

In the experiment of timestamp validation, IPv4 packet (with identification value ***52707***) sent from the client device with Unix time 1539268282 (***0x5BBF5EBA***) is dropped because the Unix time of client device is not in the range of acceptable time difference (±1 second). The experimental results are shown in figure 4.6 to 4.8.

Figure 4.6 Packet manipulation program output displayed at the client device

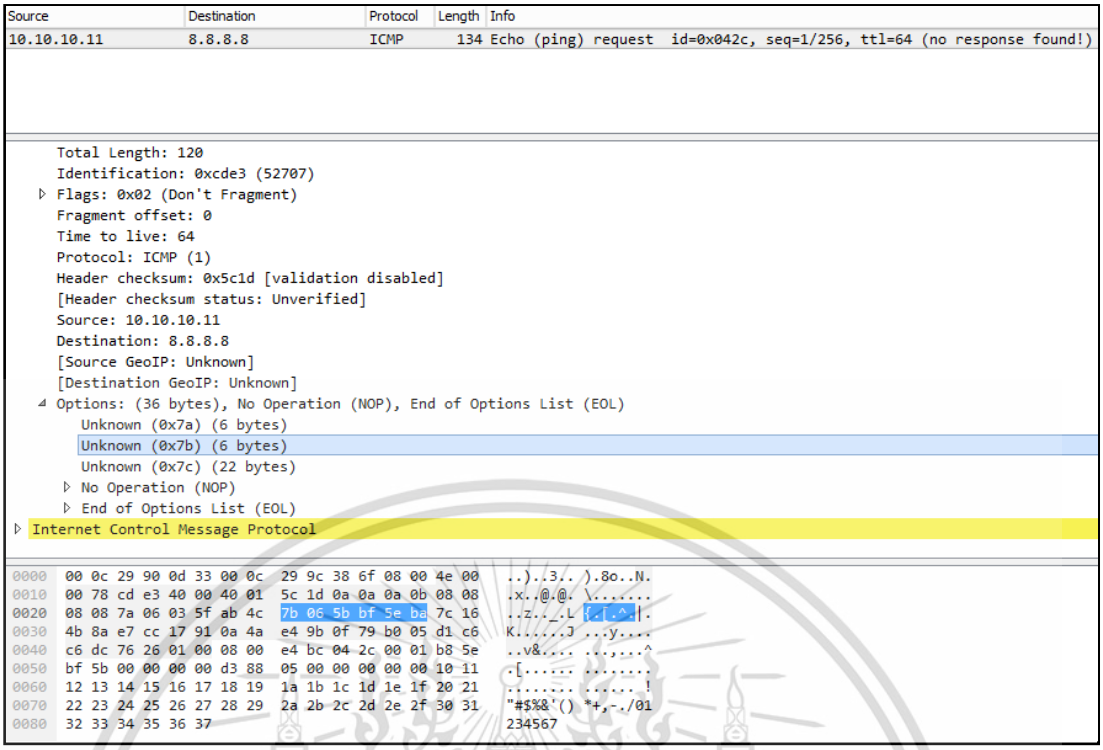


Figure 4.7 The detail of captured packet on the analyzer 2

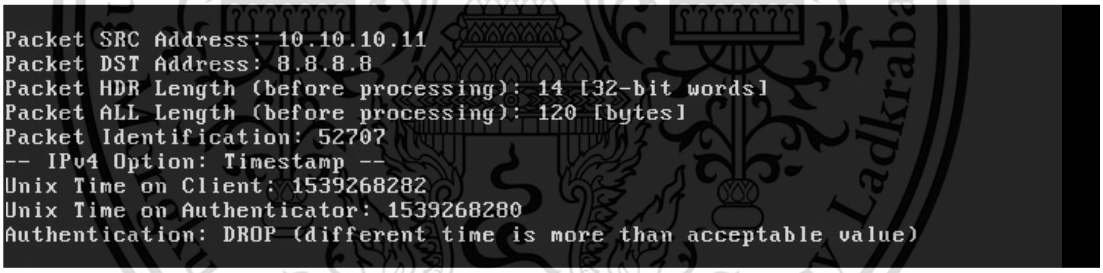


Figure 4.8 Packet manipulation program output displayed at the authenticator

In the experiment of secret key management, IPv4 packet (with identification value ***31807***) sent from the client device with user identification value 56601429 (***0x035FAB55***) is dropped at the authenticator because there is no secret key related with this user identification data in the database. The experimental results are shown in figure 4.9 to 4.11.

```
Packet SRC Address: 10.10.10.11
Packet DST Address: 8.8.4.4
Packet HDR Length (before processing): 5 [32-bit words]
Packet ALL Length (before processing): 84 [bytes]
Packet Identification: 31807
User ID Number: 56601429 [Integer], 035fab55 [Fixed HEX]
User Unix Time: 1539269236 [Integer], 5bbf6274 [Fixed HEX]
Packet AUTH Data: 84cd13d4b931b53e2a81398fb87d346444353e38 [HMAC-SHA1]
Packet HDR Length (after processing): 14 [32-bit words]
Packet ALL Length (after processing): 120 [bytes]
```

Figure 4.9 Packet manipulation program output displayed at the client device

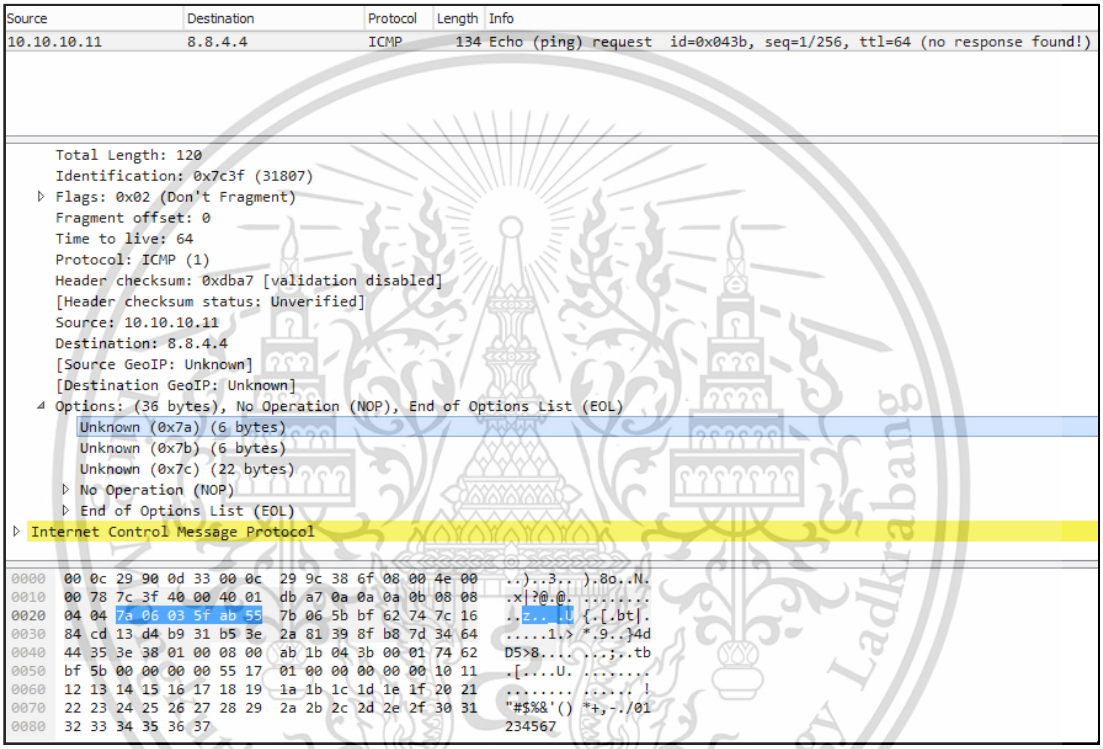


Figure 4.10 The detail of captured packet on the analyzer 2

```
Packet SRC Address: 10.10.10.11
Packet DST Address: 8.8.4.4
Packet HDR Length (before processing): 14 [32-bit words]
Packet ALL Length (before processing): 120 [bytes]
Packet Identification: 31807
-- IPv4 Option: Timestamp --
Unix Time on Client: 1539269236
Unix Time on Authenticator: 1539269235
-- IPv4 Option: User Identification --
User Identification: 56601429
Requesting key from external database
Authentication: DROP (there is no key associated with this user)
```

Figure 4.11 Packet manipulation program output displayed at the authenticator

Experimental results of packet authentication are shown in figure 4.12 to 4.18.

```
Packet SRC Address: 10.10.10.11
Packet DST Address: 8.8.8.8
Packet HDR Length (before processing): 5 [32-bit words]
Packet ALL Length (before processing): 84 [bytes]
Packet Identification: 49322
User ID Number: 56601420 [Integer], 035fab4c [Fixed HEX]
User Unix Time: 1539271069 [Integer], 5bbf699d [Fixed HEX]
Packet AUTH Data: 5f5544fb7c6c8bcf8d06722cd7d8e12ffb1e0f1b [HMAC-SHA1]
Packet HDR Length (after processing): 14 [32-bit words]
Packet ALL Length (after processing): 120 [bytes]
```

Figure 4.12 Packet manipulation program output displayed at the client device

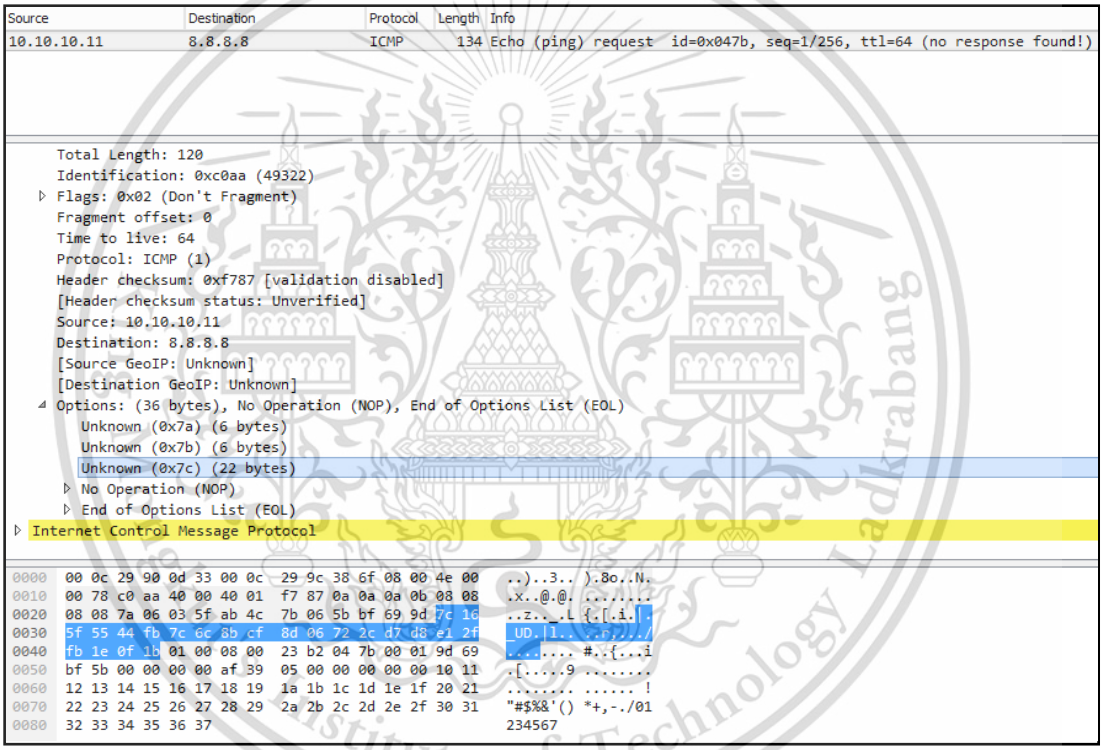


Figure 4.13 The detail of captured packet on the analyzer 2

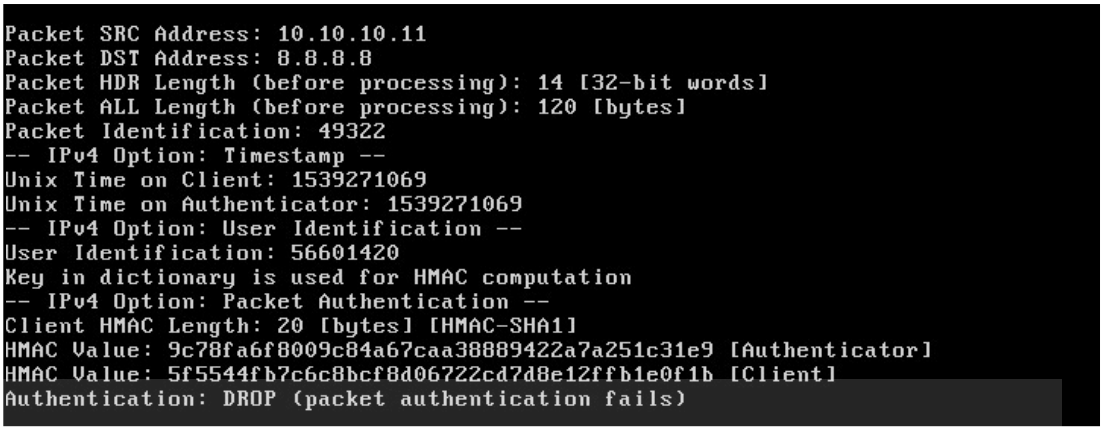


Figure 4.14 Packet manipulation program output displayed at the authenticator

As shown in figure 4.12 to 4.14, IPv4 packet (with identification value ***49322***) sent from the client device is dropped at the authenticator. The HMAC value of both sides does not match. This may due to there is a modification of data or there is an incorrect HMAC computation.

In case of sending IPv4 packet with options containing correct data in the conditions of sessionless authentication network access control. The experimental results are as follows.

Figure 4.15 Packet manipulation program output displayed at the client device

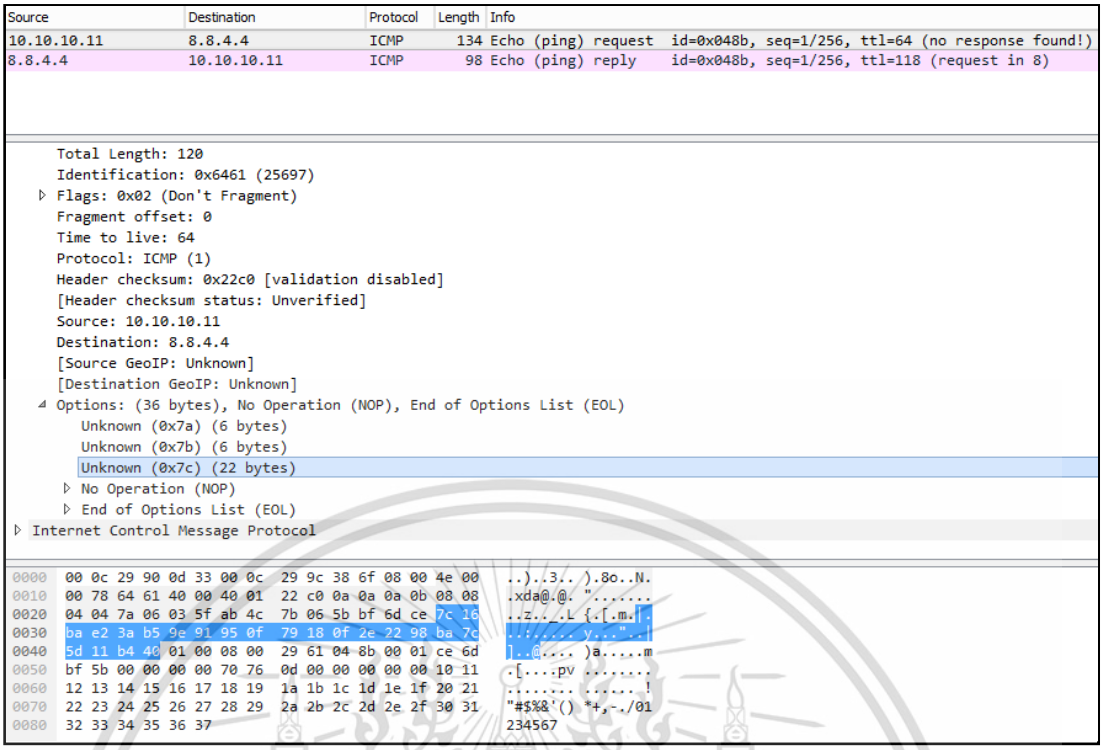


Figure 4.16 The detail of captured packet on the analyzer 2

Figure 4.17 Packet manipulation program output displayed at the authenticator

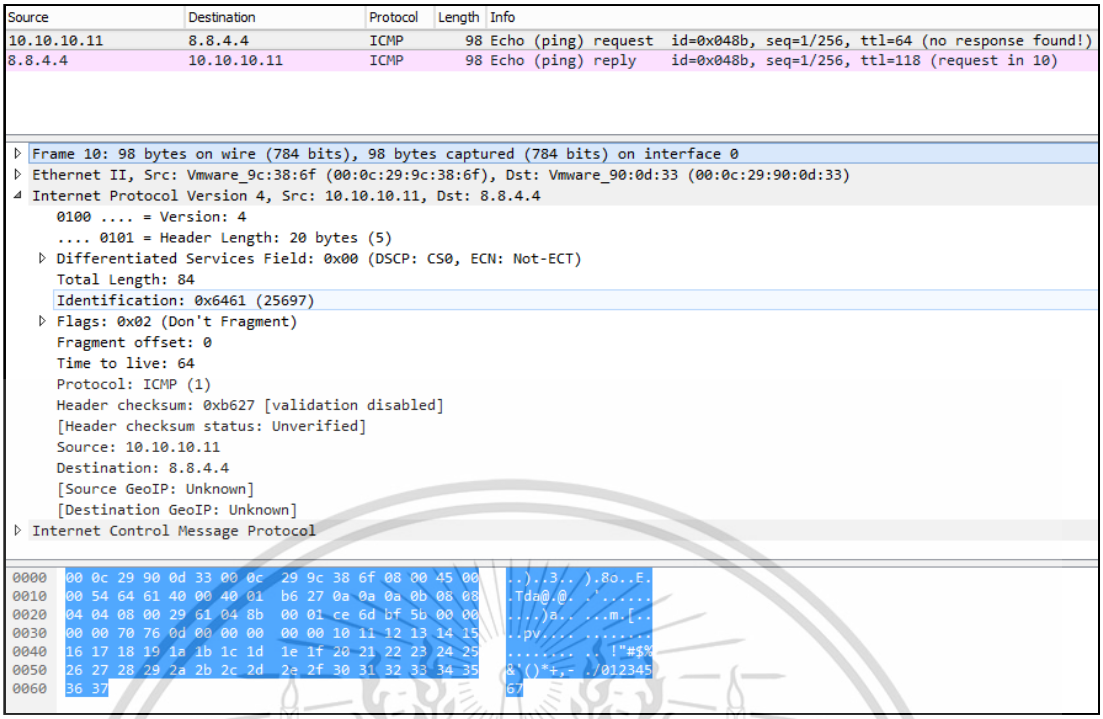


Figure 4.18 The detail of captured packet on the analyzer 1

As shown in figure 4.15 to 4.18, IPv4 packet (with identification value ***25697***) sent from the client device is allowed access to the controlled networks (the Internet). After passing the packet verification process, all attached options are removed and forwarded to the connected physical network.

4.2 Experimental Results of Routing Mode Network Access Control

In routing mode of sessionless authentication network access control, the experimental results of general packet transmission, IPv4 option numbers validation, timestamp validation, secret key management, and packet authentication are similar in case the packets are dropped. In case the packet is authenticated and authorized, the experimental results are shown in figure 4.19 to 4.22.

```
Packet SRC Address: 10.10.11.11
Packet DST Address: 8.8.4.4
Packet HDR Length (before processing): 5 [32-bit words]
Packet ALL Length (before processing): 84 [bytes]
Packet Identification: 47867
User ID Number: 56601420 [Integer], 035fab4c [Fixed HEX]
User Unix Time: 1539279947 [Integer], 5bbf8c4b [Fixed HEX]
Packet AUTH Data: 27046d49e77901d0d7ab82cea1b6425d [HMAC-MD5]
Packet HDR Length (after processing): 13 [32-bit words]
Packet ALL Length (after processing): 116 [bytes]
```

Figure 4.19 Packet manipulation program output displayed at the client device

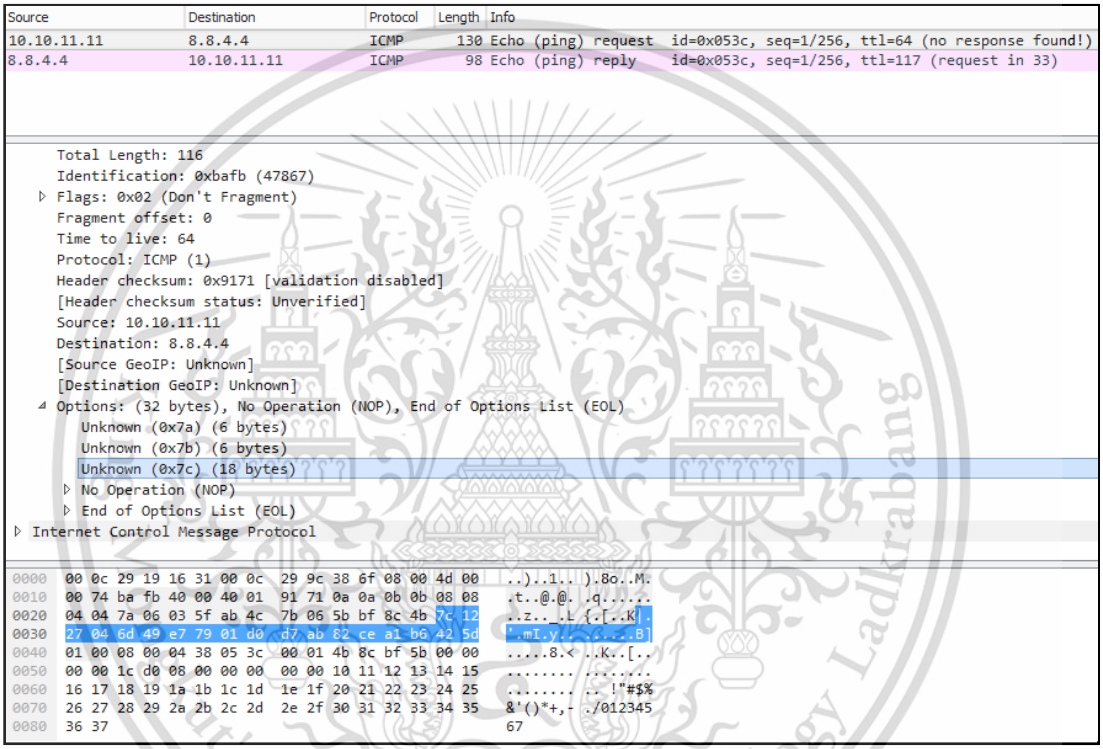


Figure 4.20 The detail of captured packet on the analyzer 2

```
Packet SRC Address: 10.10.11.11
Packet DST Address: 8.8.4.4
Packet HDR Length (before processing): 13 [32-bit words]
Packet ALL Length (before processing): 116 [bytes]
Packet Identification: 47867
-- IPv4 Option: Timestamp --
Unix Time on Client: 1539279947
Unix Time on Authenticator: 1539279947
-- IPv4 Option: User Identification --
User Identification: 56601420
Key in dictionary is used for HMAC computation
-- IPv4 Option: Packet Authentication --
Client HMAC Length: 16 [bytes] [HMAC-MD5]
HMAC Value: 27046d49e77901d0d7ab82cea1b6425d [Authenticator]
HMAC Value: 27046d49e77901d0d7ab82cea1b6425d [Client]
Authentication: ACCEPT
Packet HDR Length (after processing): 5 [32-bit words]
Packet ALL Length (after processing): 84 [bytes]
```

Figure 4.21 Packet manipulation program output displayed at the authenticator

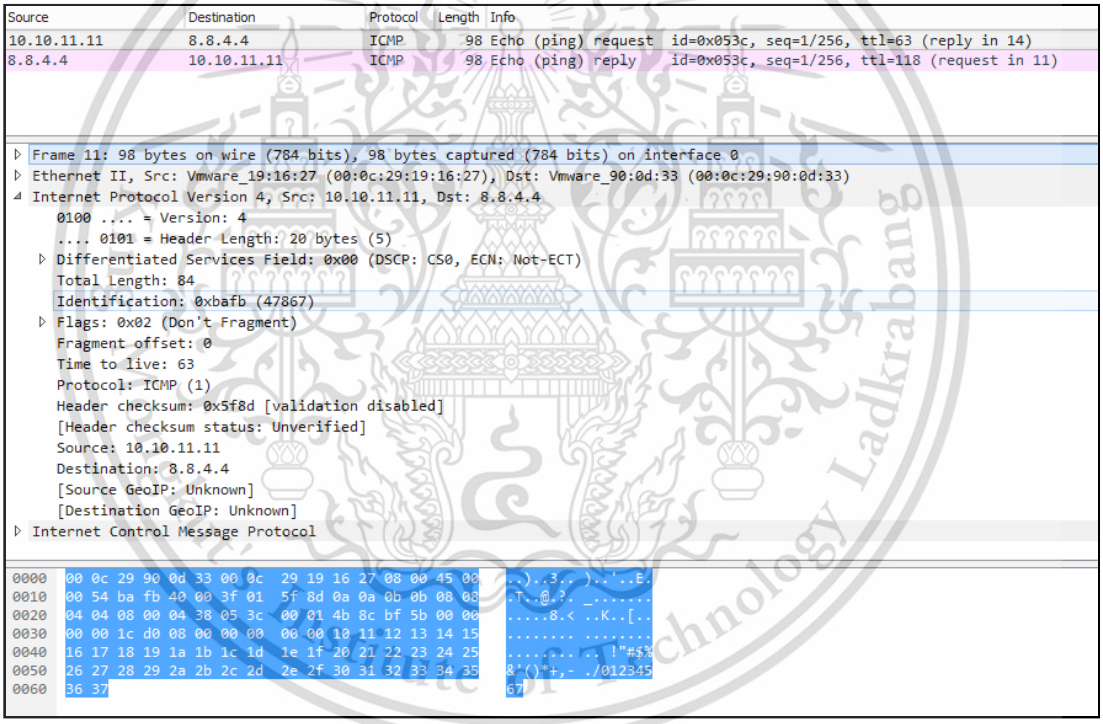


Figure 4.22 The detail of captured packet on the analyzer 1

IPv4 packet (with identification value ***47867***) sent from the client device is allowed access to the controlled networks (the Internet). After passing the packet verification process, all attached options are removed and forwarded to the connected physical network. There is a result differed from bridge mode is that the Time-to-Live value is reduced because of the routing process.

4.3 Experimental Results of Performance

In term of performance, the RTT (Round-Trip Time) is used as a performance indicator in the experiment. RTT data is collected by sending/receiving ICMP (Internet Control Message Protocol) packets between client device and NAT router. There are two experimental situations: situation 1, the sessionless authentication is deactivated, and situation 2, the sessionless authentication is activated. Results of performance testing are shown in figure 4.23, 4.24, and 4.25.

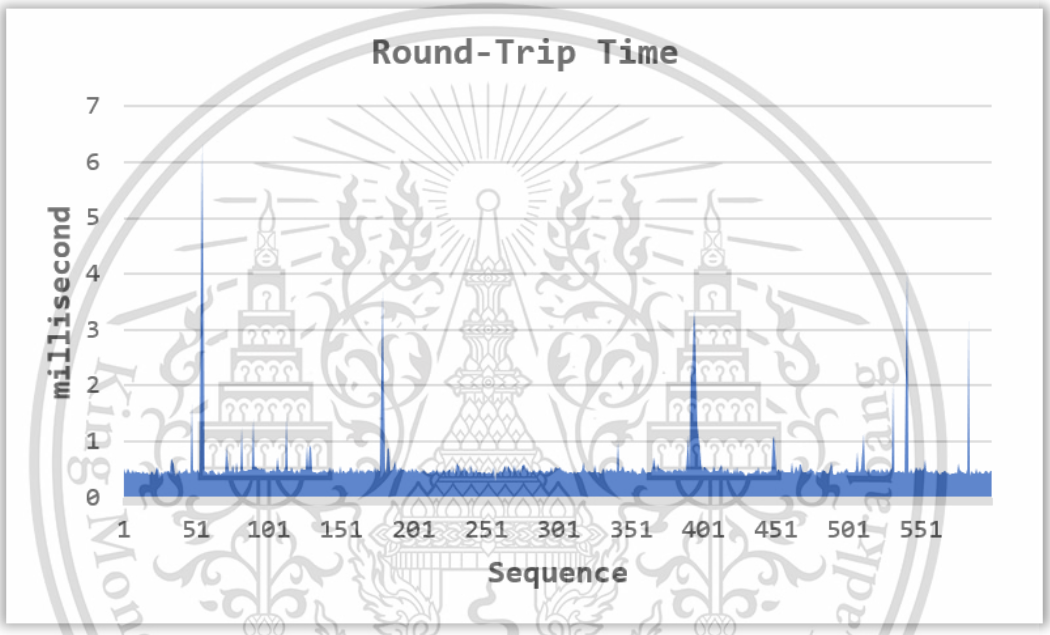
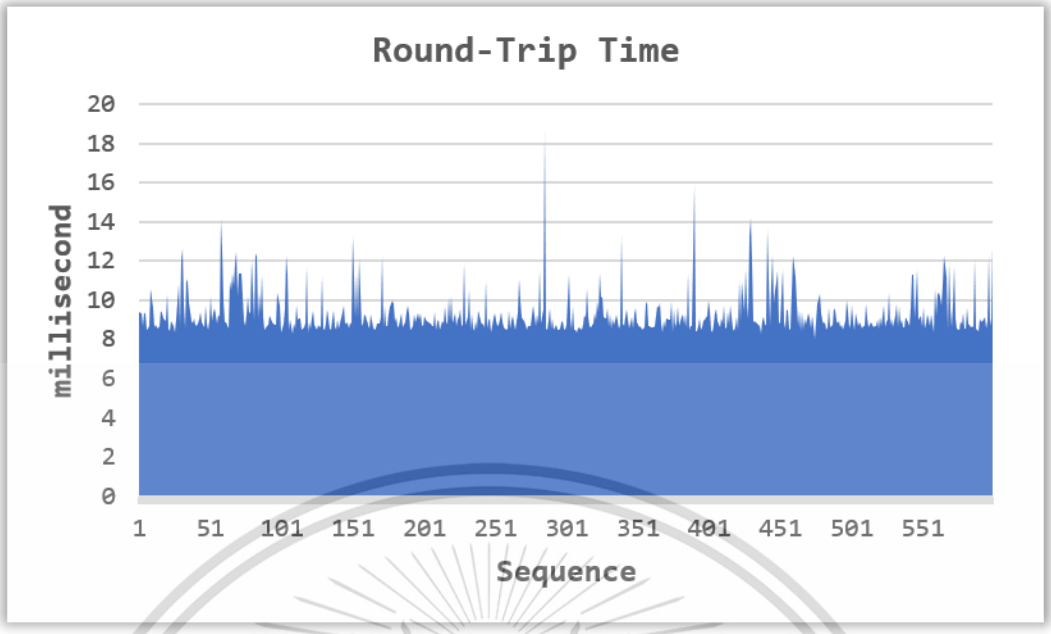


Figure 4.23 Result of ICMP RTT when the sessionless authentication is deactivated




```
600 packets transmitted, 600 received, 0% packet loss, time 300185ms
rtt min/avg/max/mdev = 8.033/9.318/18.671/1.107 ms
```

Figure 4.24 Result of ICMP RTT when the sessionless authentication is activated

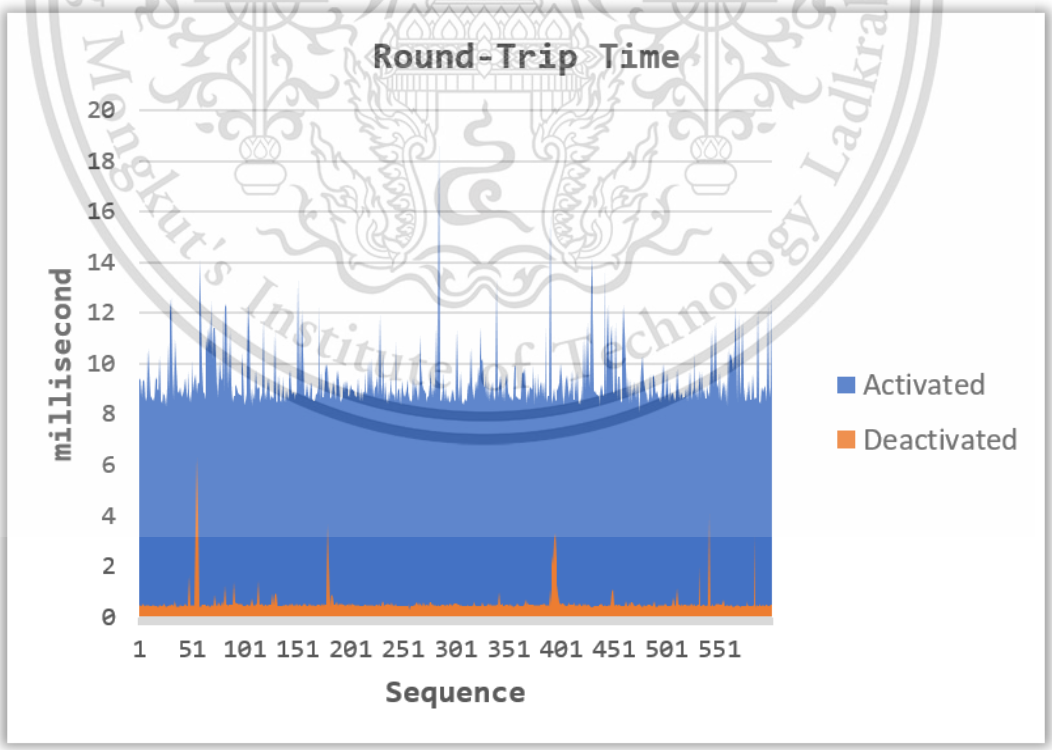


Figure 4.25 ICMP RTT results comparison chart

4.4 Discussion

All illustrated experimental results can be described that IPv4 packets sent from client device are forwarded to its destinations by the authenticator only when important IPv4 options are attached to those packets. Unique packet authentication data (HMAC) is computed at client device in a manner of packet per packet before sending the packet to the connected network, it is consistent with a mechanism of sessionless authentication. All IPv4 options must have information as they are defined to use in the sessionless authentication environment, otherwise the authenticator will drop the IPv4 packet with invalid option. In case of replay attack, IPv4 packet sent from client device may be copied and sent again within the network, the timestamp (Unix time) can be used to mitigate this attack, the authenticator will drop the replayed packet having timestamp that is not in the acceptable time tolerant. If there is a modification of data (such as timestamp, payload, value of fields of the IPv4 header, and so on), the IPv4 packet will be dropped in the process of HMAC comparison because the modified packet will result different HMAC value when recomputing the HMAC at the authenticator. IPv4 packet with option of unknown user identification data is also dropped due to the authenticator cannot find the secret key associated with unknown user identification data, if there is no secret key, HMAC computation process will not be performed at the authenticator. Sessionless authentication network access control is able to perform on both bridge and routing mode of IPv4 packet transmission and support both of HMAC-SHA1 (results in bridge mode) and HMAC-MD5 (results in routing mode). About routing process, the Time-to-Live value of IPv4 packet header is decreased, the reason is to prevent IPv4 packet from traveling on the network too long, IPv4 packet with zero value of Time-to-Live will be automatically dropped at the router. Finally, sessionless authentication network access control increases the RTT (can be seen as network latency) in the communication path which the effect of packet manipulation process.

CHAPTER 5

CONCLUSION AND SUGGESTION

An implementation of sessionless authentication in IPv4 network access control application provides the benefit of being more stringent in identifying users before allowing access to the network. The unspecified IPv4 option parameters can be used effectively in the standard IPv4 communication. The use of HMAC in conjunction with the Unix time is able to protect the source-spoofed packets or packets that are modified during the transmission and is able to mitigate the replay attack in the network. Table 5.1 shows the comparison of network access control applications between session-based authentication and sessionless authentication.

Table 5.1 Comparison of network access control applications

Features	802.1X	Captive Portal	Sessionless Authentication
Supported mode	Bridge	Bridge/Routing	Bridge/Routing
Per-packet user authentication	No	No (check only source address)	Yes (***0x7A*** option)
Replay attack mitigation	No	No	Yes (***0x7B*** option)
Data integrity validation and data origin authentication	No	No	Yes (***0x7C*** option)
Require additional packet manipulation process	No	No	Yes (using Scapy and Netfilter)
Packet overhead	No	No	MD5, +32 bytes SHA1, +36 bytes

As shown in the table, although the sessionless authentication has more overhead, but when considering in authentication performance and security, the features of sessionless authentication make the user identification more precise and it is difficult for unauthorized users in attempting to send packets to the Internet.

In term of development guidelines for sessionless authentication network access control to be more effective in the future, there are two possible issues that can be developed. Firstly, the ability of replay attack mitigation might be upgraded from the level of mitigation to the level of protection. Secondly, the IPv4 option parameters might be inherited to use with IPv6 communication in part of extension headers.

REFERENCES

[1] G. Appenzeller, M. Roussopoulos, and M. Baker, “User-Friendly Access Control for Public Network Ports,” INFOCOM IEEE, Vol. 2, March 1999, pp. 699-707.

[2] “802.1X-2001 IEEE Standard for Local and Metropolitan Area Networks Port-Based Network Access Control,” IEEE, 2001.

[3] C. Manusankar, S. Karthik, and T. Rajendran, “Intrusion Detection System with Packet Filtering for IP Spoofing,” International Conference on Communication and Computational Intelligence, December 2010, pp. 563-567.

[4] K. Jan and H. Christian, “A Secure Token-Based Communication for Authentication and Authorization Servers,” International Conference on Future Data and Security Engineering, October 2016, pp. 237-250.

[5] H. Krawczyk, M. Bellare, and R. Canetti, “HMAC: Keyed-Hashing for Message Authentication,” RFC 2104, February 1997.

[6] R. Rivest, “The MD5 Message-Digest Algorithm,” RFC 1321, April 1992.

[7] D. Eastlake and P. Jones, “US Secure Hash Algorithm 1 (SHA1),” RFC 3174, September 2001.

[8] Wikipedia, “Cryptographic hash function,” [Online], Available: https://en.wikipedia.org/wiki/Cryptographic_hash_function, 2018.

[9] Wikipedia, “HMAC,” [Online], Available: https://en.wikipedia.org/wiki/HMAC, 2018.

[10] M. Heather, B. Satish, S. Oleg, and T. Rohit, “Practical Mobile Forensics,” 3rd ED, Packt Publishing Limited, January 2018.

[11] D. Denning and G. Sacco, “Timestamps in Key Distribution Protocols,” Communications of the ACM, Vol. 24, August 1981, pp. 533-536.

[12] P. Syverson, “A taxonomy of replay attacks,” IEEE Computer Society Press, 1994, pp. 187-191.

[13] A. Behrouz Forouzan, “Data Communications and Networking,” 5th ED, McGraw-Hill, 2013.

[14] J. Postel, “INTERNET PROTOCOL,” RFC 791, September 1981.

[15] S. Deering and R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification,” RFC 2460, December 1998.

[16] Wikipedia, “IPV4,” [Online], Available: https://en.wikipedia.org/wiki/IPv4, 2018.

REFERENCES (Cont.)

[17] Internet Assigned Numbers Authority (IANA), “Internet Protocol Version 4 (IPv4) Parameters,” [Online], Available: https://www.iana.org/assignments/ip-parameters/ip-parameters.xhtml, 2018.

[18] L. Michael, B. Philippe, B. Nicolas, M. Eric, G. Paul, B. Bryan, M. Eric, I. Chris, G. Jennifer, M. Steve, K. Dave, and S. Julien, “Security Power Tools,” O'Reilly Media Inc, August 2007.

[19] R. Brandon and G. John, “Foundations of Python Network Programming,” 3rd ED, Apress, 2014.

[20] G. Lucian, “Designing and Implementing Linux Firewalls and QoS using netflter, iproute2, NAT, and L7-flter,” Packt Publishing Limited, October 2006.

APPENDIX

A Design of Self-Authenticated Internet Protocol for Network Access Control

Parinya Thamthawornsakul
Department of Telecommunication Engineering
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
56601420@kmitl.ac.th

Suvepon Sittichivapak
Department of Telecommunication Engineering
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
kssuvepo@kmitl.ac.th

***Abstract*—This paper presents an application of Internet Protocol Options for enhancing self-authenticated feature to the Internet Protocol. The options consist of Keyed-Hash Message Authentication Code and timestamp. In the network access control system, local network devices needing to access protected networks have to attach the options to packet header before sending to destination through a network access controller or commonly known as a gateway. Internet Protocol packets with options are verified on the gateway, the Keyed-Hash Message Authentication Code is used for checking integrity of data and the timestamp is used for replay attack mitigation. Packets passing all processes would be allowed to access to the protected networks. The design of self-authenticated Internet Protocol provides a protection of sending source spoofed packets from local network causing damage to other networks and also provides high reliability of source identification.**

Keywords—Internet Protocol Options; Network Access Control; Source Spoofing; HMAC

I. INTRODUCTION

NAC (Network Access Control) is a part of computer network security that attempts to verify an identity of the source before allowing to access to destination network. In order to verify the identity, an authentication is highly imperative. Currently, Captive Portal technique and IEEE 802.1x standard are authentication methods for controlling access to networks and being widely used in many places such as academies, organizations, Etc.

Captive Portal has offered the first use in the Stanford's SPINACH project [1]. It is a web-based authentication service. This technique focuses on the user-friendly network access control. Captive Portal is used at many Wi-Fi hotspots, and can be able to control wired network access as well. Despite this technique provided convenient network access control, it is susceptible to source address spoofing. It is vulnerability that malefactors can access protected network easily without passing the authentication process.

The IEEE 802.1x is a standard defined for port-based network access control. It provides the authentication mechanism to client devices needed to connect to the network. The IEEE 802.1x consists of three components; a supplicant, an authenticator, and an authentication server. A supplicant is a client network device needed to connect to any network. An authenticator is a network device, such as an Ethernet switch, wireless access point performing like a security guard for any protected network. Usually a supplicant is not allowed to access to the protected network through the authenticator until the supplicant's identity has been validated and authorized by the authentication server. The IEEE 802.1x normally provides a highly secure authentication mechanism, especially to any wireless network. However, it cannot prevent the traffic manipulated by source address spoofing on the wired network [2]. This problem occurs the same as it does with Captive Portal network access control.

Source address spoofing is the creation of frames or packets with forged source addresses, for the purpose of concealing the sender's identify or masquerading as another computing system. Data communication, being based on the protocol not supporting the origin authentication, cannot detect and prevent traffic generated by this technique. Hence it is a problem without an easy solution [3].

The aforementioned problems can be solved by replacing the currently used network access control methods with an implementation of self-authenticated IP (Internet Protocol). Local network devices needing to access protected networks have to attach specific IP options to IP header before sending IP packet to destination through a gateway. The IP options consist of HMAC (Keyed-Hash Message Authentication Code) and timestamp. The HMAC is calculated by using some important fields in IP header, higher layer data in IP packet, timestamp, and key as input parameters. At the gateway, IP packets with options are verified, the new HMAC is calculated by using the same parameters as the senders, and is compared to the attached HMAC, IP packets with HMAC and timestamp corresponding to the new HMAC and timestamp on gateway would be allowed to pass through gateway to the protected network. With using the self-authenticated IP, source address spoofing traffic can be detected and protected. Moreover, the local network device or network user identification could be considered as highly reliable.

978-1-4673-9749-0/16/$31.00 ©2016 IEEE

II. Literature Review

The design of self-authenticated IP is the application of several theories which are described as follows.

A. Internet Protocol and Options

Internet Protocol is the principal communication protocol for carrying datagrams across network boundaries. It has the task of delivering datagrams from the source host to the destination host, based on the IP addresses in the IP headers. The IP standard is described in RFC (Request for Comments) 791 [4], the IP header format is shown in figure 1.

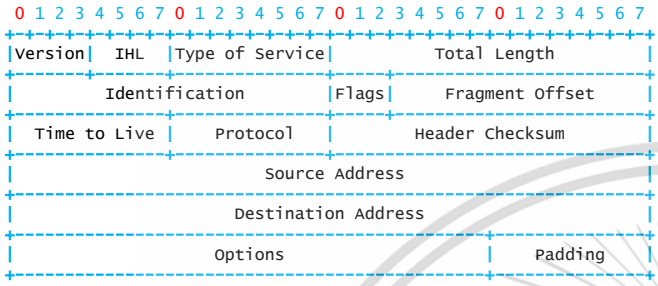


Fig. 1. Internet Protocol header format.

In part of IP options, it is a field in the IP header which may be required in some communication environments. The option field is variable in length. There are two cases for the format of an option; a single octet of option-type, and multiple octets of option-type. The single octet of option-type indicates the end of the option list or it may be used between options, to align the beginning of a subsequent option on a 32-bit boundary. The multiple octets of option-type can be divided into three parts as shown in figure 2.

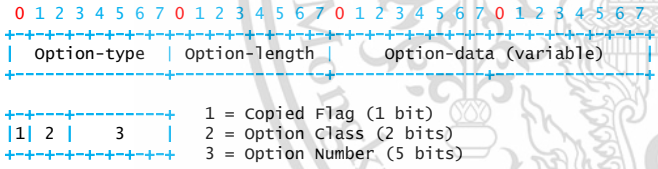


Fig. 2. Internet Protocol option field format.

According to figure 2, the option-type octet is viewed as having three fields; 1 bit for copied flag, 2 bits for option class, and 5 bits for option number. The copied flag indicates that this option is copied into all fragment IP packets on fragmentation, the option is not copied when use value 0, and the option is copied when use value 1. The option class indicates the general category into which the option belongs, there are only four values for option class; class 0 for control, class 2 for debugging and measurement, class 1 and class 3 are reserved for future use. The option number specifies the kind of option, there are some numbers defined in the RFC and there are also inactive numbers. For the last two parts of the option field, the option-length octet counts the option-type octet and the option-length octet as well as the option-data octets, and the option-data octets contain the actual data of each IP option in the option field.

B. Keyed-Hash Message Authentication Code (HMAC)

In order to check the integrity of information transmitted over unreliable networks, mechanisms that provide the integrity check are often called HMAC [5]. HMAC is a mechanism for message authentication using cryptographic hash functions. HMAC can be used with any iterative cryptographic hash function in combination with a secret shared key. Most commonly used hash functions are MD5 [6] and SHA-1 [7]. Equation (1) describes the definition of HMAC.

$$HMAC(K, m) = H((K \oplus opad) \,||\, H((K \oplus ipad) \,||\, m)) \quad (1)$$

Regarding the equation, H is a cryptographic hash function, K is a secret shared key, m is the message to be authenticated, $||$ donates concatenation, $\oplus$ donates exclusive or (XOR), *opad* is the outer padding, and *ipad* is the inner padding. The operation of generating HMAC is illustrated in figure 3.

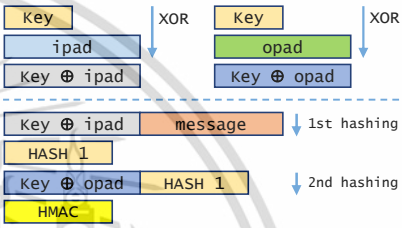


Fig. 3. Generating the HMAC.

The iterative hash function breaks up a message into blocks of a fixed size and iterates over them with a one-way compression function. For example, MD5 and SHA-1 operate on 512-bit blocks. The size of the output of HMAC is 128 bits in the case of MD5 and 160 bits in the case of SHA-1. The size of a secret shared key must be less than or equal to the block size of each hash function. The strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size of its output, and on the size and quality of the secret shared key.

C. UNIX Timestamp

The UNIX timestamp is a system time defined as the number of seconds that have elapsed since 00:00:00 UTC (Coordinated Universal Time), Thursday, 1 January 1970. For example, timestamp number 1451017392 represents 04:23:12 UTC, Friday, 25 December 2015. The standard UNIX timestamp data-type that represents a point in time is a signed integer, traditionally of 32 bits, meaning that it covers a range of about 136 years in total. The maximum representable time is 03:14:07 UTC, Tuesday, 19 January 2038. One second after that maximum time would be overflow. The timestamp is widely used in UNIX-like and many other operating systems and file formats.

In data security, timestamp is combined with a hashing mechanism to guarantee that the data has not been modified during transmission. The sender and the receiver use HMAC generated by actual data, secret shared key, and timestamp to ensure that transmitted data is not corrupted or modified. The

timestamp is used, along with some threshold, to ensure that particular packet cannot be used more than once, for example, using timestamp in [8]. In replay attack mitigation, if the threshold (acceptable time lag, in seconds) is small, then the replay attack is almost impossible.

III. Design and Experimentation

A. Self-Authenticated IP Option Field Format

In the design of option field format for self-authenticated IP, the reserved option classes and the inactive option numbers are used. Two new IP options is inserted in the IP header before sending IP packet out, one is an option containing a HMAC to be used to authenticate IP packet itself, and the other one is an option containing a timestamp to be used to prevent replay attack. The self-authenticated IP option field format is illustrated in figure 4.

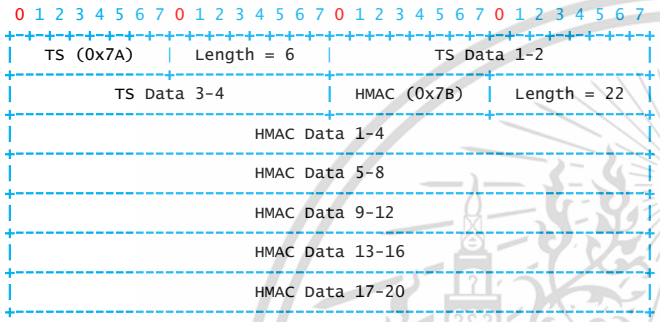


Fig. 4. IP option field format for self-authenticated IP.

In figure 4, the timestamp (TS) option uses hexadecimal number 0x7A for option-type, which is 01111010 in binary number, meaning that this option would not be copied into all fragments on fragmentation (0), this option is in the reserved option class 3 (11), and this option uses inactive option number 26 (11010). The length of the timestamp option is 6 octets including 1 octet of option-type, 1 octet of option-length, and 4 octets of timestamp data. The timestamp data is a hexadecimal number converted from represented number of the UNIX timestamp.

The HMAC option uses 0x7B for option-type having the same properties as the timestamp option, but the slight difference is an option number, which is an inactive number 27 (11011). The length of the HMAC option is 22 octets including 1 octet of option-type, 1 octet of option-length, and 20 octets of calculated HMAC SHA-1 (160 bits). In part of calculating the HMAC SHA-1, an IHL field, a Total Length field, an Identification field, a Protocol field, a Source Address field, and a Destination Address field in the IP header, higher layer data (IP payload), and timestamp are concatenated and used as input parameter of the hash function. The secret shared key is one of input parameters separately. It is used both on a network device and a gateway, the source address in an IP header is a pointer to the same secret shared key.

B. Traffic Flow and Operations

The self-authenticated IP is implemented on an open-source operating system, which is Linux. In term of system operations, it can be viewed into two parts, which are kernel space and user space. The kernel space provides abstraction for security, hardware, and internal data structures. The user space refers to all of the code in an operating system that lives outside of the kernel, such as shells and applications. About network traffic flow on Linux, it can be operated in both kernel space and user space, normally network traffic is operated in the kernel space, but it can also be operated in the user space, depending on the security policy.

The implementation uses Netfilter framework to bridge IP traffic between kernel space and user space via the Netfilter queue (NFQUEUE), to modify IP options in the IP header by the specific program running in user space, before sending the modified IP packets back to the kernel space and then to the network interface. This operation can be illustrated as shown in figure 5.

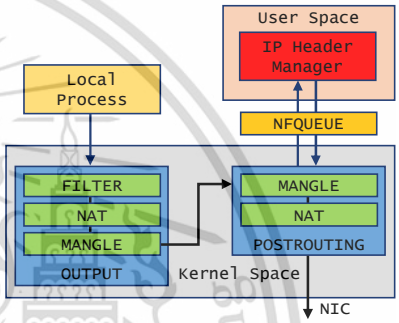


Fig. 5. Self-Authentication IP traffic flow on local network device.

In the figure 5, the NFQUEUE is used to bridge outgoing IP packets generated by local network device at Mangle table, POSTROUTING chain. This operation is controlled by the Netfilter's IPTable command is as follows.

```
[1] iptables -t mangle -A POSTROUTING -o <out_if> ! -d <int_net> -j
NFQUEUE --queue-num <q_number>
```

The IP packets destined other than local network (*int_net*) are bridged to user space via NFQUEUE (*q_number*, start with number 0 or higher). Then, the modified IP packets are sent back to kernel space and sent on wire by an outgoing network interface (*out_if*).

At the gateway, for incoming IP traffic, IP packets destined other than local network (*int_net*) are received by an incoming network interface (*in_if*) and are bridged to user space via NFQUEUE (*q_number_A*) at Mangle table, PREROUTING chain. In the user space, IP packets are verified, and accepted or dropped by an IP header manager program. Accepted IP packets are sent back to kernel space and are routed to the destination, therefore these IP packets will become outgoing IP packets. For outgoing traffic, IP packets are bridged to user space again via NFQUEUE (*q_number_B*). The IP options are removed and sent back to kernel space. Finally, authenticated IP packets are sent on wire by an outgoing network interface (*out_if*). The IPTable commands are as follows.

```
[1] iptables -t mangle -A PREROUTING -i <in_if> ! -d <int_net> -j
NFQUEUE --queue-num <q_number_A>
[2] iptables -t mangle -A POSTROUTING -o <out_if> -j NFQUEUE --
queue-num <q_number_B>
```

Traffic management at the gateway can be illustrated as shown in figure 6.

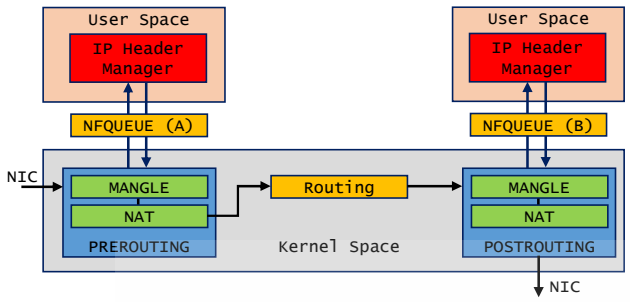


Fig. 6. Self-Authentication IP traffic flow on gateway.

In a part of inserting, removing, or modifying IP options in the IP header, Scapy [9] is one of useful tools for this operation. Scapy is a powerful interactive packet manipulation program written in Python. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. In the implementation, the pseudo code of IP option insertion is as follows.

```
[1]  def callback(i, payload):
[2]        data = payload.get_data()
[3]        pkt = IP(data)
[4]        pkt[IP].options = IPOption("%s%s%s%s"%("\x7A\x06",
ts_data, "\x7B\0x16", hmac_data))
[5]        pkt[IP].ihl += 7
[6]        pkt[IP].len = len(str(pkt))
[7]        del pkt[IP].chksum
[8]        payload.set_verdict_modified(nfqueue.NF_ACCEPT, str(pkt),
len(pkt))
[9]  def main():
[10]       q = nfqueue.queue()
[11]       q.open()
[12]       q.bind(socket.AF_INET)
[13]       q.set_callback(callback)
[14]       q.create_queue(0)
[15]       try:
[16]                 q.try_run()
[17]       except KeyboardInterrupt:
[18]                 q.unbind(socket.AF_INET)
[19]                 q.close()
[20] main()
```

In the code, the *ts_data* variable contains the current timestamp, and the *hmac_data* variable contains the calculated HMAC. Firstly, traffic flowing in kernel space are captured and sent to user space by using an *nfqueue.queue* function. In the callback function, the *pkt* variable contains IP packet extracted from the *payload* variable. The *pkt[IP].options* is a pointer that points the IP option field position in IP header. New IP options are inserted in a form of *char* data-type, including the *ts_data* and the *hmac_data*. The length of the new inserted IP options is 7 words (1 word is equal to 32 bits), therefore the IHL value in the IP header is increased, from 5 words to 12 words (5+7). Finally, the values of total length and header checksum fields are recalculated, before sending (reinjecting the modified IP header to the kernel space) to destinations.

For HMAC calculation, given an *icv* variable contains calculated HMAC, a *pkt* variable contains IP packet data, a *ts* variable contains timestamp data in hexadecimal value, and a *key* variable contains secret shared key based on source IP address of each IP packet. The procedure for calculating the HMAC can be written as shown in the following pseudo code.

```
[1]  def gen_icv(pkt, ts, key):
[2]        ip_pkt = str(pkt).encode("hex")
[3]        icv = hmac.new(key,"",hashlib.sha1)
[4]        icv.update(str(pkt[IP].ihl+7))
[5]        icv.update(str(pkt[IP].len+(7*4)))
[6]        icv.update(str(pkt[IP].id))
[7]        icv.update(str(pkt[IP].proto))
[8]        icv.update(str(pkt[IP].src))
[9]        icv.update(str(pkt[IP].dst))
[10]       icv.update(ip_pkt[pkt[IP].ihl*4*2:])
[11]       icv.update(str(ts))
[12]       return icv.digest()
```

Input parameters used for calculating the HMAC SHA-1 consist of some data fields in IP packet, the *pkt[IP].ihl* represents the Internet Header Length data, the *pkt[IP].len* represents the Total Length data, the *pkt[IP].id* represents the Identification data, the *pkt[IP].proto* represents Protocol data, the *pkt[IP].src* represents the Source Address data, the *pkt[IP].dst* represents the Destination Address data, the *ip_pkt[pkt[IP].ihl*4*2:]* represents data in higher layer, and timestamp data.

For incoming packet processing at the gateway, IP header fields are classified and stored in variables of the IP header manager program similar to outgoing packet processing. After receiving IP packet, the timestamp option is checked and the HMAC SHA-1 is recalculated and compared with the HMAC option in the received packet. The process of IP packet verification is described as a flowchart shown in figure 7.

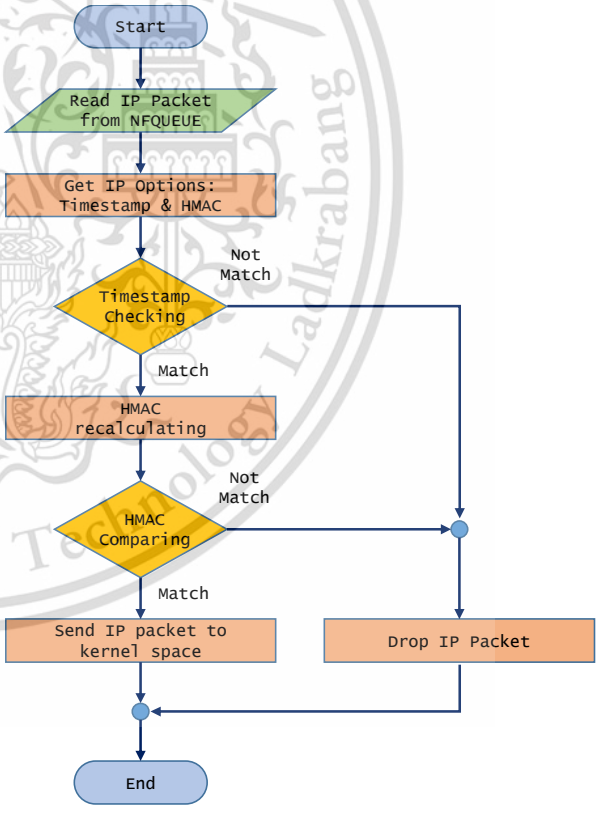


Fig. 7. IP packet verification process.

In the packet verification process, firstly, timestamp data in the timestamp option is checked with timestamp on a gateway.

IP packets with timestamp does not correspond to timestamp of the gateway would be dropped. Secondly, IP packets passing the timestamp checking are brought into the HMAC calculation process. HMAC data in the HMAC option is compared to the new calculated HMAC on the gateway. IP packets with HMAC does not correspond to the new HMAC would be dropped. IP packets passing all verification processes are sent back to the kernel space. For outgoing traffic, accepted IP packets are sent to the IP header manager again, self-authentication IP specific options are removed before sending the IP packets to network interface.

C. *Experimental Environment*

Because of the self-authenticated IP is implemented on Linux. The experimental environment requires few personal computers, for running IP header manager programs and capturing sample IP packets transmitting over Ethernet network. Detail of the experimental environment is shown in figure 8.

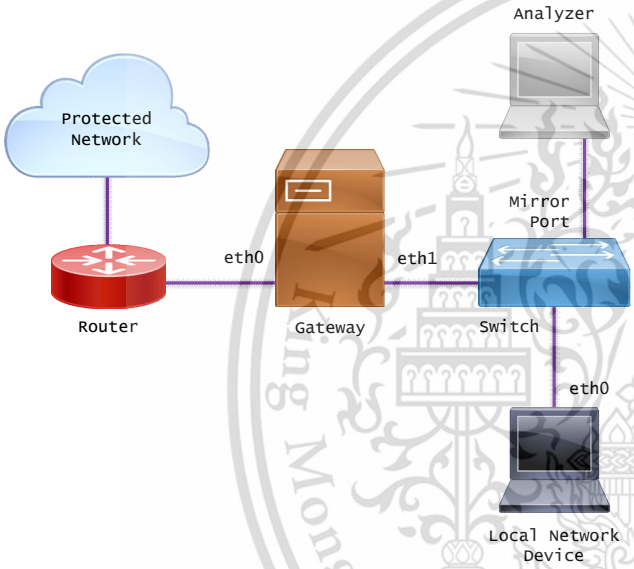


Fig. 8. Experimental environment for self-authenticated IP.

In the experiment, an *eth0* interface on local network device is connected with a managed switch. In order to communicate with the protected network, local network device need to send IP packets to a gateway (Network Access Controller), the packets are received by *eth1* interface of the gateway. IP packet that passing the timestamp checking and the HMAC comparing are routed to the protected network via *eth0* interface of the gateway. The managed switch copies IP packets generated by local network device to other port connected with an analyzer. IP packets captured by the analyzer are used as an experimental result. Network configurations in the test environment are as follows:

1) Local Network Device:

a) eth0 IP Address: 172.16.0.2/24

b) Default Route: 172.16.0.1

c) DNS Server: 8.8.8.8

2) Gateway (Network Access Controller):

a) eth1 IP Address: 172.16.0.1/24

b) eth0 IP Address: 10.0.0.2/30

c) Default Route: 10.0.0.1

d) DNS Server: 8.8.8.8

3) Router:

a) Local IP Address: 10.0.0.1/30

b) Global IP Address: Assigned by Service Provider

c) Default Route: Assigned by Service Provider

d) DNS Server: Assigned by Service Provider

The experiment is a proof of self-authenticated IP security concept. The results are described in the next section.

IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

During the experiment, the self-authenticated IP enabled device communicated with the protected network, by using ICMP (Internet Control Message Protocol). The *ping* program is executed on the local network device with destination IP address 8.8.4.4. The analyzer captured related IP packets. On the analyzer console, the specific IP options is discovered and displayed in the sub-tree menu as shown in figure 9.

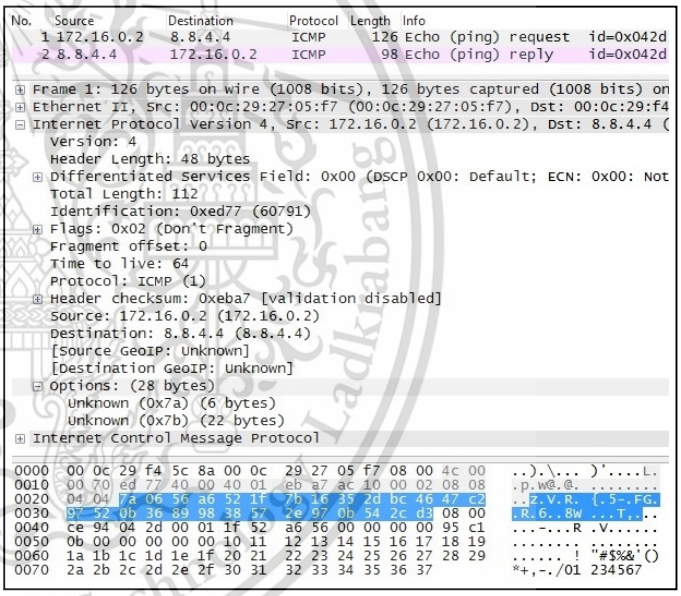


Fig. 9. Specific IP options displayed on an analyzer console.

The analyzer classified the specific IP options into two options, but there is not a detail for each option because the self-authenticated IP options are non-standard IP options defined in the RFC. But it confirms that the specific IP options are really added in the IP header before sending IP packet out on a wire.

At the gateway, in part of incoming packet processing, when add the code *pkt[IP].show()* in the IP header manager program, it allows the program to display a detail of the specific IP options. The detail of the options of the same IP packet captured by the analyzer is shown in figure 10.

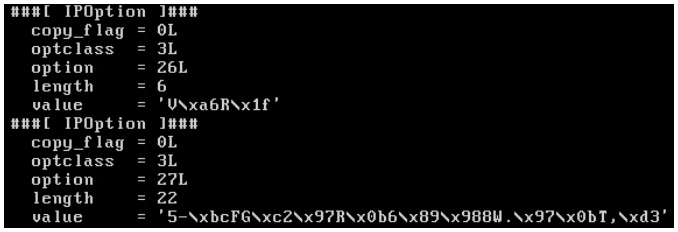


Fig. 10. Specific IP options displayed on Linux console (Scapy).

In the detail, each option has option-type, option-length, and option-data corresponding to the design of option field format. In term of security, the IP header manager program on local network device has been modified to simulate sending some unusual IP packets to the gateway. The conditions and results of the simulation are list in table 1.

TABLE I. IP PACKET VERIFICATION TEST RESULT

Item	Condition	Accepted	Dropped
1	Sending IP packet with IP options corresponding to timestamp and HMAC.	•	
3	Timestamp option on local network device is increased/decreased before calculating the HMAC.		•
4	Using different secret shared key for calculating the HMAC of the same IP packet.		•
5	Input parameters for calculating the HMAC does not meet the specifications.		•
6	Sending IP packet to protected network without adding the self-authenticated IP options in the IP header.		•

Information in the table can be ensured that source spoofed packets or packets that have been modified during transport are unable to reach their destination, because it is dropped when checked at the gateway.

V. CONCLUSIONS

The self-authenticated IP provided high security for Network Access Control. With the design based on IP options, it can be implemented with the standard IP communication without any effect. In the real implementation, the secret shared key can be mapped with a user and a source IP address. Therefore, accessing to external network or protected network with a strict inspection would make high accuracy in user identification, in case a problem occurs and requires to check. In the overall, self-authenticated IP is a lightweight method to prevent an unauthorized access to external networks using the source address spoofing, which is the major problem in controlling network access.

REFERENCES

[1] G. Appenzeller, M. Roussopoulos, and M. Baker, "User-Friendly Access Control for Public Network Ports," INFOCOM IEEE, vol. 2, March 1999, pp. 699-707.

[2] A. Duckwall, "Defeating Wired 802.1x with a Transparent Bridge Using Linux," unpublished.

[3] C. Manusankar, S. Karthik, and T. Rajendran, "Intrusion Detection System with Packet Filtering for IP Spoofing," International Conference on Communication and Computational Intelligence, India, December 2010, pp. 563-567.

[4] J. Postel, "INTERNET PROTOCOL," RFC 791, September 1981.

[5] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104, February 1997.

[6] R. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321, April 1992.

[7] D. Eastlake and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," RFC 3174, September 2001.

[8] D. Denning and G. Sacco, "Timestamps in Key Distribution Protocols," Communications of the ACM, Vol. 24, August 1981, pp. 533-536.

[9] P. Biondi, "Packet generation and network based attacks with Scapy," CanSecWest/core05, France, May 2005.

2017 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)

An Implementation of Internet Protocol Options for Self-Authentication

Parinya Thamthawornsakul
Department of Telecommunication Engineering
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
56601420@kmitl.ac.th

Suvepon Sittichivapak
Department of Telecommunication Engineering
King Mongkut's Institute of Technology Ladkrabang
Bangkok, Thailand
kssuvepo@kmitl.ac.th

***Abstract*—This paper presents an enhancement of IP (Internet Protocol) standard to support user authentication within the protocol itself. The options field in an IP header is used for carrying specific data to add the ability of self-authentication. The specific data consist of a user identifier, a timestamp, and an HMAC calculated with important data in the IP header. The major purpose is to verify a device owner or a computer user in a local network in real time, before allowing access to restricted networks or the Internet. By this enhancement, the users can be authenticated at IP layer, without needing an additional user authentication process. The self-authentication ability provides a prevention of sending source-spoofed IP packet and also provides a high reliability of identifying the user. In addition, this ability does not require a creation of specific connection and an exchange of security parameters.**

Keywords—Network Access Control; Source Address Spoofing; IP Options; HMAC; Self-Authentication

I. INTRODUCTION

Nowadays, Internet almost become a basic requirement of daily life. A lot of devices online on the Internet. Regarding computer network security, identifying a device owner or a computer user is an importance. In order to control accessing the Internet, a NAC (Network Access Controller) is used to meet this demand. On an IP network, the NAC provides a function of source address filtering. IP packets with a source address of an authorized device are allowed to access the Internet. Although NAC is a useful tool for identifying the user before accessing restricted networks or the Internet, but its common function is susceptible to a technique of sending a source-spoofed IP packet, it helps unauthenticated users easily access their destination without passing the user authentication process.

The aforementioned problem encourages an enhancement of IP standard. A self-authentication ability is added to the protocol. It involves a use of undefined IP options, and an application of HMAC (Keyed-Hash Message Authentication Code) and timestamp. The benefits of this enhancement are user authentication can be performed at IP layer without needing additional processes, sending the source-spoofed IP packet can be prevented, and reliability of identifying the user is increased.

II. LITERATURE REVIEW

A. Source Address Spoofing

In term of computer network security, the creation of IP packet with a forged source address for concealing the sender identity, is called source address spoofing. Source-spoofed IP packets travelling on an IP network cannot be detected and protected. It may cause damage to various services on the network. Most importantly, source address spoofing is a problem without an easy solution [1]. Traditional NAC such as Captive Portal [2] still use the source address filtering function to control accessing restricted networks, hence it will consider the source-spoofed IP packet as a normal IP packet. Both types of IP packet are finally allowed to access the restricted networks.

The source address spoofing issue may be mitigated by using a communication tunnel. In this scenario, tunnel is a transport protocol for carrying the regular protocol. The tunnel is designed for providing data origin authentication and data integrity verification. These interesting functions make more reliable communication at the IP layer.

B. Origin Authentication and Integrity Verification

Data origin authentication is an important property. It makes the receiving party can verify the source of data and ensure that the IP packet originate from an expected sender. Data integrity verification is also important one that makes sure about IP packet has not been modified during transmission. Using the two properties, the source-spoofed IP packet can be detected and protected.

There are several protocols providing data origin authentication and data integrity verification. Most commonly used protocol is an IPsec (Internet Protocol Security) [3], in a functionality of AH (Authentication Header) [4]. However, IPsec needs extended authentication process, a communication session need to be maintained, and it is a host-to-host security. But for NAC application, it should be host-to-any security, the data origin authentication and the data integrity verification should be performed without maintaining any session or state.

C. Stateless Authentication Mechanism

An access control system with session maintenance, the period of communication depends on the session. If the session expires, communication will stop and require re-authentication

978-1-5386-0449-6/17/$31.00 ©2017 IEEE

to continue. In the other hand, the stateless authentication mechanism does not maintain any communication session. There is an agent program installed on a device or a computer. It attaches sufficient user credentials to the IP packet, the credentials is used for verifying the device owner on the NAC, this mechanism may be called self-authentication.

Creating the credentials for stateless user authentication, the HMAC [5] is commonly used. HMAC is a mechanism for message authentication. It can be used with any iterative cryptographic hash function, in a combination of message (data) and secret key. The HMAC definition is described in equation 1.

$$HMAC(K, m) = H((K \oplus opad) \,||\, H((K \oplus ipad) \,||\, m)) \quad (1)$$

In the equation, H is a cryptographic hash function, K is a secret key, m is the message to be authenticated, || represents concatenation, $\oplus$ is an XOR digital logic gate, *opad* is an outer padding, and *ipad* is an inner padding. The strength of the HMAC depends on the strength of underlying cryptographic hash functions such as a SHA-1 (Secure Hash Algorithm) [6], the size of its output, and the quality of the secret key.

Using the HMAC for stateless authentication mechanism may not secure enough because it is susceptible to a replay attack [7]. To reduce the possibility of the attack, a timestamp is required. The sender and the receiver use HMAC with timestamp to guarantee that the IP packet has not been modified during transmission and cannot be used more than once. An example case has been described in [8]. The timestamp is often used with threshold (acceptable time lag), if the threshold is small, the replay attack is almost impossible.

D. Internet Protocol Options

IP options is one of fields in the IP header [9] which may be required for some data communication environments. The IP options field is variable in length. There are two types of the options; a single octet of option-type, and multiple octets of option-type. The single octet of option-type indicates the end of the IP options list or it may be used between IP options, to align the beginning of a subsequent option on a 32-bit boundary. The multiple octets of option-type can be divided into three parts as shown in figure 1.

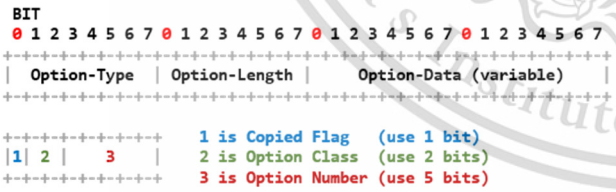


Figure 1. Multiple octets of option-type format

Regarding the figure 1, the option-type octet is viewed as having three parts; 1 bit for copied flag, 2 bits for option class, and 5 bits for option number. The copied flag indicates that this option is copied into all fragment IP packets on fragmentation, the option is not copied when using a value 0, or is copied when using a value 1. The option class indicates the general category into which the option belongs, there are only four option classes; class 0 for control, class 2 for debugging and measurement, class 1 and class 3 are reserved for future use. The option number specifies the kind of option, there are both defined option numbers in the IP standard and undefined numbers. The option-length octet indicates how many octets are used in each option. It counts the option-type octet, the option-length octet as well as the option-data octets. The last option-data octets contain the actual data of each option.

III. Implementation of User Authentication in an Internet Protocol

A. Self-Authentication IP Options

In the design of self-authentication IP options, the reserved option classes and the undefined option numbers are used. Three new IP options are attached to the IP header; a user identifier option, a timestamp option, and an HMAC option, which can be illustrated in figure 2.

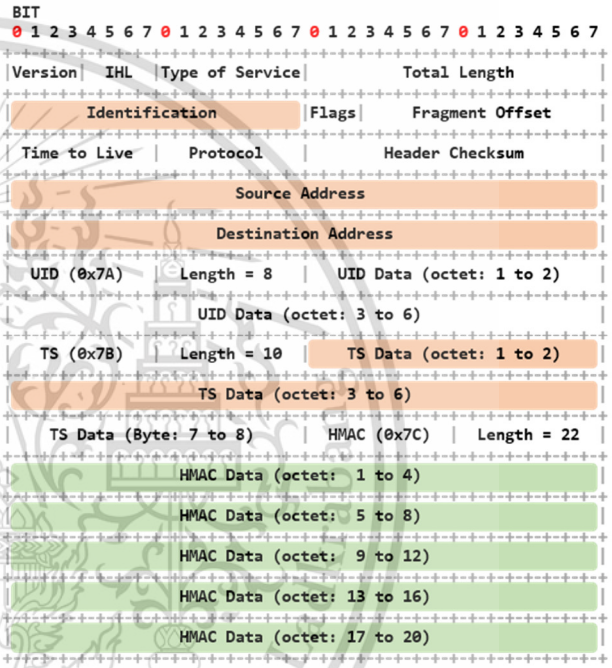


Figure 2. Self-Authentication options

In the options format, the user identifier option (UID) has value 0x7A, that is 01111010 in binary number, so that this option is not copied to all fragments on fragmentation (0), the reserved option class 3 (11) is used for this option, and the option number 26 (11010) is assigned for this option. The total length in octet is 8, including 1 octet of option-type, 1 octet of option-length, and 6 octets of the user identifier data. NAC will use this option to find the corresponding secret key for generating the HMAC. The timestamp option (TS) has value 0x7B. It has property like the UID option, except using option number 27. The total length in octet is 10, including 1 octet of option-type, 1 octet of option-length, and 8 octets of the timestamp data which is a hexadecimal number converted from an integer number of the UNIX timestamp. The timestamp is used to compare with the timestamp of the NAC for replay attack mitigation purpose. The last HMAC option has value 0x7C. It has property like the UID option, except using option

number 28. The total length of this option is 22, including 1 octet of option-type, 1 octet of option-length, and 20 octets of HMAC data calculated with the SHA-1 hash function (160 bits of output).

In HMAC calculation, the Identification data, the Source Address data, the Destination Address data in the IP header, the timestamp in the TS option, and the secret key corresponding to the UID option, are concatenated and used as an input message of the hash function.

B. *Data Flow and Operations*

The self-authentication is implemented on Linux. In order to control the flow of IP packet and manipulate the IP header, Netfilter framework and Scapy [10] program are applied. The Netfilter framework provides IP traffic control on the experiment computers. The *POSTROUTING* chain *mangle* table of the framework is mainly used to do mangling on IP packets. At this chain and this table, the IP packets are redirected from kernel space to user space by *NFQUEUE* (the Netfilter Queue). Scapy is a powerful interactive IP packet manipulation program written in Python. It works on the user space, and it can inject the IP packet back to the kernel space.

The self-authentication ability focuses on an outgoing IP traffic. In the implementation, IP packet flow on both client device and NAC are shown in figure 3 and 4.

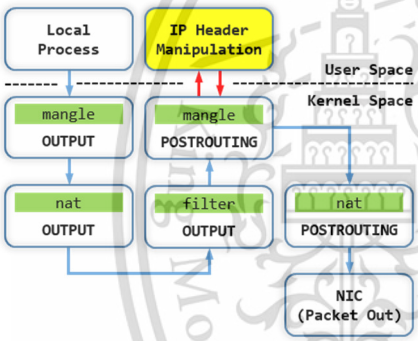


Figure 3. IP packet flow on client device

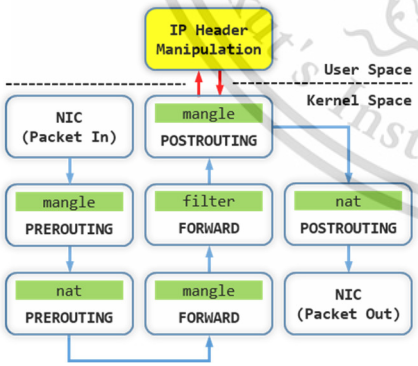


Figure 4. IP packet flow on NAC

On client device, the IP header manipulation program attaches the UID option, the TS option, and the HMAC option respectively, to IP header generated by local process. The IHL value in the IP header is increased to 15 (1111), as well as the Header Checksum value is recalculated. Finally, the modified IP packet is injected back to the kernel space and sent out to the network. At the NAC, IP packets with self-authentication options are verified and forwarded to another network. An action on IP packet on the NAC is described in figure 5.

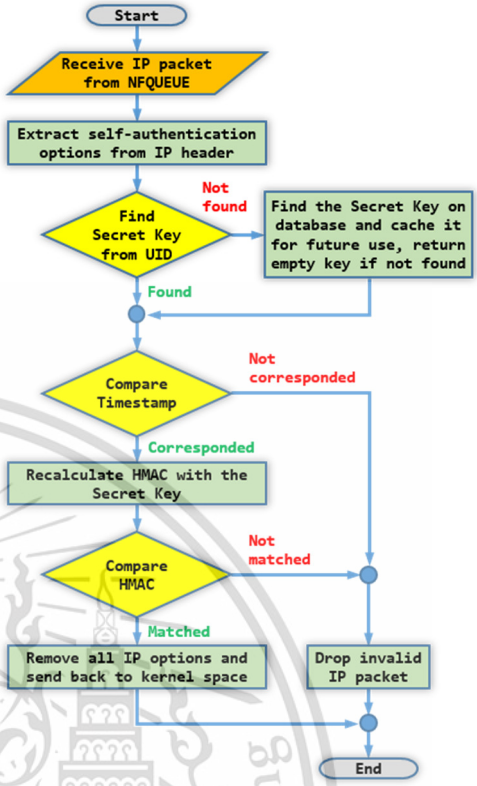


Figure 5. Self-Authentication verification

IV. Experimentation and Results

The experimental environment consists of four computers, two network switches, and one Internet router. The computers work as a client device, a NAC, and analyzers. The test IP packet will be sent from the client device. IP packets passing the network switches are mirrored to the analyzers. The captured IP packets on the analyzer are used as an experimental results. The network topology is shown in figure 6.

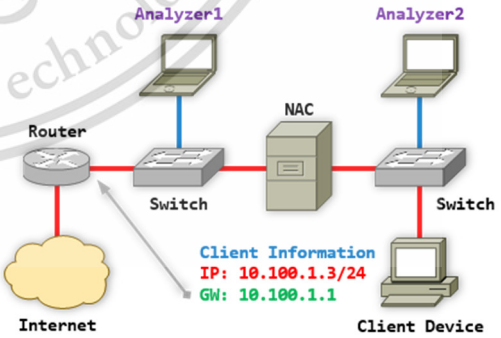


Figure 6. Experimental topology

The client device has one NIC (Network Interface Card), named *ens160*. The NAC has two NICs, named *ens160* (router side) and *ens192* (client side), an operation is in bridge mode.

In order to redirect IP packets to the IP header manipulation program, the following *iptables* commands are executed on both client device and NAC console.

```
# # IPTables command for Client Device.
# iptables -t mangle -I POSTROUTING \
> -o ens160 -j NFQUEUE --queue-num 0

# # IPTables command for NAC.
# iptables -t mangle -I POSTROUTING \
> -m physdev --physdev-out ens160 \
> -j NFQUEUE --queue-num 0
```

After sending the test ICMP packet from the client device (10.100.1.3) to the destination (8.8.8.8), Captured IP packets on the analyzer1 and the analyzer2 are decoded as displayed in figure 7 and 8.

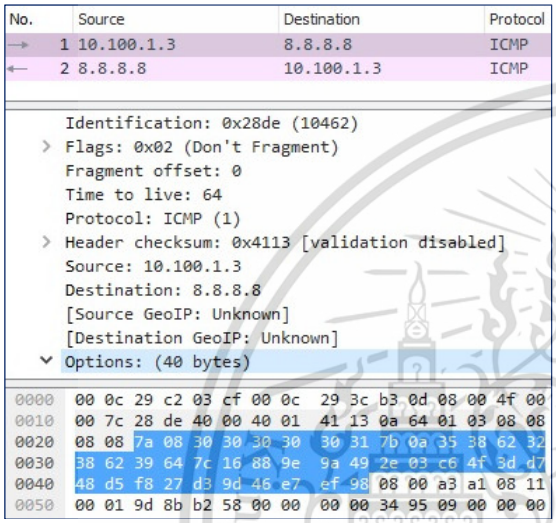


Figure 7. IP packet decoded on the analyzer1

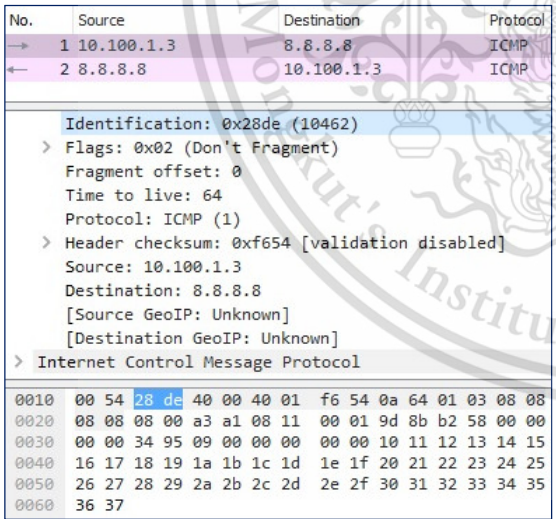


Figure 8. IP packet decoded on the analyzer2

IP packet with identification number 10462 is sent from client device, the self-authentication options are attached correctly as described in the section of implementation. After passing verification on the NAC, self-authentication options are deleted and the IP packet is forwarded to another network side. In addition, when considering about the mentioned network access control, the key features of each access control application can be compared and shown in table 1.

TABLE I. A COMPARISON OF ACCESS CONTROL APPLICATIONS

Features \ Access Control	IP options	IPsec AH	Captive Portal
Access control ability	Yes	Yes	Yes
Source-spoofed IP packet prevention	Yes	Yes	No
Data origin authentication and data integrity verification	Host-to-Any	Host-to-Host	No
Not require session maintenance	Yes	No	No
Not require security parameter exchange	Yes	No	N/A

V. CONCLUSION

The implementation provides a new mechanism for verifying an identity of the device owner or the computer user. It is especially suitable to use with network access control application. The key feature is a design of self-authentication options. User authentication at IP layer can be performed by NAC which can be placed anywhere on the communication path. Without needing the additional user authentication process and there is no session to be maintained, it makes data communication more smooth and seamless. For security awareness, it also provides a mitigation of replay attack and a prevention of sending the source-spoofed IP packet, therefore it make identifying the user more reliable and it makes user authentication mechanism look different.

REFERENCES

[1] C. Manusankar, S. Karthik, and T. Rajendran, "Intrusion Detection System with Packet Filtering for IP Spoofing," International Conference on Communication and Computational Intelligence, India, December 2010, pp. 563-567.

[2] G. Appenzeller, M. Roussopoulos, and M. Baker, "User-Friendly Access Control for Public Network Ports," INFOCOM IEEE, vol. 2, March 1999, pp. 699-707.

[3] S. Kent and K. Seo, "Security Architecture for the Internet Protocol," RFC 4301, December 2005.

[4] S. Kent, "IP Authentication Header," RFC 4302, December 2005.

[5] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104, February 1997.

[6] D. Eastlake and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," RFC 3174, September 2001.

[7] P. Syverson, "A taxonomy of replay attacks," IEEE Computer Society Press, 1994, pp. 187-191.

[8] D. Denning and G. Sacco, "Timestamps in Key Distribution Protocols," Communications of the ACM, Vol. 24, August 1981, pp. 533-536.

[9] J. Postel, "INTERNET PROTOCOL," RFC 791, September 1981.

[10] P. Biondi, "Packet generation and network based attacks with Scapy," CanSecWest/core05, France, May 2005.

User Authentication in an Internet Protocol

Parinya Thamthawornsakul Suvepon Sittichivapak
Department of Telecommunication Engineering, King Mongkut's Institute of Technology Ladkrabang

Abstract

This paper presents an enhancement of IP (Internet Protocol) standard to support user authentication within the protocol itself. The options field in an IP header is used for carrying specific data to add the ability of self-authentication. The specific data consist of a user identifier, a timestamp, and an HMAC calculated with important data in the IP header. The major purpose is to verify a device owner or a computer user in a local network in real time, before allowing access to restricted networks or the Internet. By this enhancement, users can be authenticated at IP layer, without needing an additional user authentication process. The self-authentication ability provides a prevention of sending source-spoofed IP packet and also provides a high reliability of identifying the user. In addition, this ability does not require a creation of specific connection and an exchange of security parameters.

Keywords: IP Options, Self-Authentication, HMAC, Source Address Spoofing, Network Access Control

1. Introduction

Nowadays, Internet almost become a basic requirement of daily life. A lot of devices online on the Internet. Regarding computer network security, identifying a device owner or a computer user is an important thing. In order to control accessing the Internet, a NAC (Network Access Controller) is used to meet this demand. On an IP network, the NAC provides a function of source address filtering. IP packets with a source address of an authorized device are allowed to access the Internet. Therefore prior to such authorization, the device owner or the computer user has to pass the authentication process first, which is done by submitting user credentials such as a username and a password.

Although NAC is a useful tool for identifying the user before accessing restricted networks or the Internet, but its common function is susceptible to a technique of sending a source-spoofed IP packet. This technique helps an unauthenticated user easily access restricted networks or the Internet, without passing any user authentication process.

The aforementioned problem encourages an enhancement of IP standard. A self-authentication ability is added to the protocol. It involves a use of undefined IP options, and an application of HMAC (Keyed-Hash Message Authentication Code) and timestamp. The benefits of this enhancement are user authentication can be performed at IP layer without needing an additional user authentication process, sending a source-spoofed IP packet can be prevented, and reliability of identifying the user is increased.

2. Literature Review

2.1. Source Address Spoofing

In term of computer network security, the creation of IP packet with a forged source address for concealing the sender identity, is called source address spoofing. Due to the IP standard has not designed for providing data origin authentication and data integrity verification, a source-spoofed IP packet travelling on an IP network cannot be detected and protected. It may cause damage to various services on the network. Most importantly, source address spoofing is a problem without an easy solution [1].

Traditional NAC such as Captive Portal [2] still use the source address filtering function to control accessing restricted networks. An access controller monitors and filters only IP standard packet. That means NAC will consider the source-spoofed IP packet as a normal IP packet. Both of IP packets are finally allowed to access the restricted networks.

The source address spoofing issue may be mitigated by using a communication tunnel. In this scenario, tunnel is a transport protocol for carrying the regular protocol. The tunnel is designed for providing data origin authentication and data integrity verification. These interesting functions make more reliable communication at the IP layer.

2.2. Data Origin Authentication and Data Integrity Verification

In term of reliable data communication, transferring information with security awareness is significance. Data origin authentication and data integrity verification must be fully applied on the tunneling protocol.

Data origin authentication is an important property. It makes the receiving party can verify the source of data and ensure that the IP packet originate from an expected sender. Hence, the source-spoofed packet can be detected and protected. Data integrity verification is an important one that makes sure about IP packet has not been modified during transmission.

It is the assurance of accuracy and the consistency of data in the IP packet.

There are several tunneling protocols used in computer network security. Most commonly used tunneling protocol supporting both data origin authentication and data integrity verification is an IPsec (Internet Protocol Security) [3], in a functionality of AH (Authentication Header) [4]. However, the IPsec requires an extended user authentication process before establishing the tunnel, there is a session that need to maintain. This can be viewed as a complex mechanism. To suppress the complexity, data origin authentication and data integrity verification provided on the communication tunnel should be performed without maintaining any session or state.

2.3. Stateless Authentication Mechanism

In network access control system with user session, a period of data communication depends on such user session. If the user session expires, the communication will stop and require re-authentication process to continue. In the other hand, the stateless authentication mechanism does not has a session manipulation. An agent program installed on a device or a computer attaches sufficient user credentials to the IP packet, which is used to authenticate the device owner or computer user on the NAC. The mechanism may be called self-authentication.

To create user credentials for stateless authentication, HMAC [5] is commonly used to do this. HMAC is a mechanism for message authentication. It can be used with any iterative cryptographic hash function, in a combination of message (data) and secret key. The HMAC definition is described in equation 1.

$$HMAC(K,m) = H((K \oplus opad)||H((K \oplus ipad)||m)) \quad (1)$$

In the equation, H is a cryptographic hash function, K is a secret key, m is the message to be authenticated, $||$ represents concatenation, $\oplus$ is an exclusive or (XOR) digital logic gate, *opad* is an outer padding, and *ipad* is an inner padding. The order of HMAC calculation is illustrated in figure 1.

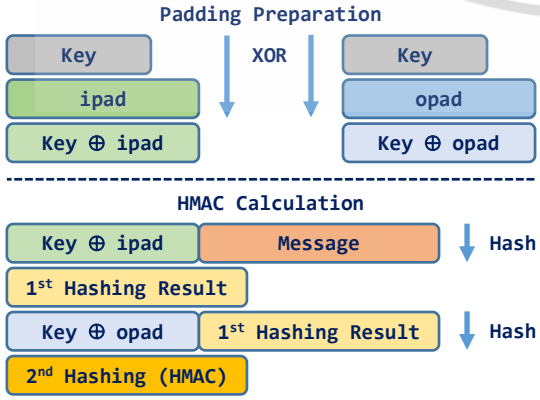


Figure 1 HMAC calculation

The iterative hash function breaks up a message into blocks of a fixed size and iterates over them with a one-way compression function. The strength of the HMAC depends on the strength of underlying cryptographic hash functions such as a SHA-1 (Secure Hash Algorithm) [6], the size of its output, and the quality of the secret key.

Using the HMAC for stateless authentication mechanism may not secure enough. The mechanism is susceptible to a replay attack [7]. By this attack, an attacker is able to duplicate the IP packet with user credentials of an authenticated user, transmit the IP packet to the same network. To reduce the possibility of the replay attack, timestamp is required.

The sender and the receiver use HMAC with timestamp to guarantee that the data is not corrupted or the data has not been modified during transmission. The timestamp is often used with some threshold to ensure that particular IP packet cannot be used more than once. An example case has been described in [8]. About replay attack mitigation, if the threshold of timestamp (acceptable time lag) is small, the replay attack is almost impossible.

2.4. Internet Protocol Options

IP options is one of fields in the IP header [9] which may be required for some data communication environments. The IP options field is variable in length. There are two option formats defined in the IP standard; a single octet of option-type, and multiple octets of option-type. The single octet of option-type indicates the end of the IP options list or it may be used between IP options, to align the beginning of a subsequent option on a 32-bit boundary. The multiple octets of option-type can be divided into three parts as shown in figure 2.

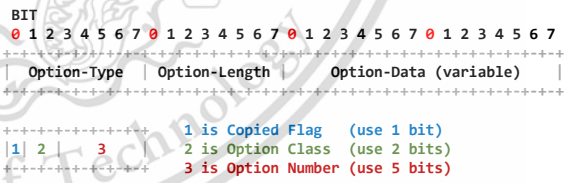


Figure 2 Multiple octets of option-type format

Regarding the figure 2, the option-type octet is viewed as having three parts; 1 bit for copied flag, 2 bits for option class, and 5 bits for option number. The copied flag indicates that this option is copied into all fragment IP packets on fragmentation, the option is not copied when using a value 0, and the option is copied when using a value 1. The option class indicates the general category into which the option belongs, there are only four option classes; class 0 for control, class 2 for debugging and measurement, class 1 and class 3 are reserved for future use. The option number specifies the kind of option, there are both defined option numbers in the IP standard and undefined numbers. The option-

length octet indicates how many octets are used in each option. It counts the option-type octet, the option-length octet as well as the option-data octets. The last option-data octets contain the actual data of its option.

3. Implementation of User Authentication in an Internet Protocol

3.1. IP Options Design for Self-Authentication

In the design about the new IP options for self-authentication ability, reserved option classes and undefined option numbers are used. Three new IP options are attached to the IP header. The first option is a user identifier, it is used as an index for finding the corresponding secret key. The next option is a timestamp, it is used in replay attack mitigation. The last option is an HMAC, it is used for data integrity verification of the IP packet itself. The new IP options can be illustrated in figure 3.

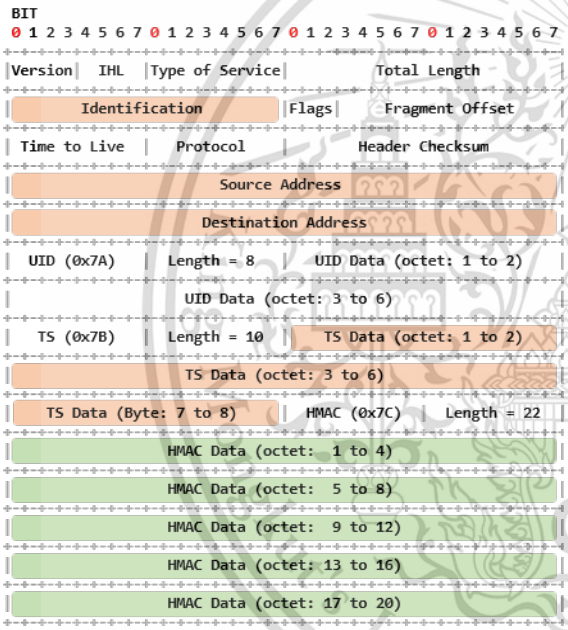


Figure 3 IP header with self-authentication options

In IP options format, the user identifier option (UID) is defined as a hexadecimal number 0x7A in option-type octet, that is 01111010 in binary number, so that this option is not copied to all fragments on fragmentation (0 value of the first bit), a reserved option class 3 (11) is used for this option, and a number 26 (11010) is assigned for this option number. The total length in octet of this option is 8, including 1 octet of option-type, 1 octet of option-length, and 6 octets of the user identifier data. NAC will use this option to find the corresponding secret key for generating the HMAC.

The timestamp option (TS) is defined as a hexadecimal number 0x7B in option-type octet, that is 01111011 in binary number, so that this option is not copied to all fragments on fragmentation, it is in the reserved option class 3, and a number 27 (11011) is assigned for this option number. The total length of this option is 10, including 1 octet of option-type, 1 octet of option-length, and 8 octets of the timestamp data which is a hexadecimal number converted from an integer number of the UNIX timestamp. The timestamp is used to compare with the timestamp of the NAC for replay attack mitigation purpose.

The last HMAC option is defined as a hexadecimal number 0x7C in option-type octet. It has the same properties as the previous two options, and a number 28 (11100) is assigned for this option number. The total length of this option is 22, including 1 octet of option-type, 1 octet of option-length, and 20 octets of HMAC data calculated with the SHA-1 hash function (160 bits of output).

In HMAC calculation, the Identification data, the Source Address data, the Destination Address data in the IP header, the timestamp data in the timestamp option, and the secret key corresponding to the user identifier option, are concatenated and used as an input message of the hash function. The procedure of HMAC calculation can be written in the following pseudo code.

```
def gen_icv(packet_in, ts_in, key_in):
  icv = hmac.new(key_in, "", hashlib.sha1)
  icv.update(str(packet_in[IP].id))
  icv.update(str(packet_in[IP].src))
  icv.update(str(packet_in[IP].dst))
  icv.update(hex(ts_in)[2:])
  return icv.digest()
```

The input message that is used for calculating the HMAC contains both of persistent data and dynamic data. The **packet_in[IP].id** represents data in the Identification field, the **packet_in[IP].src** represents data in the Source Address field, the **packet_in[IP].dst** represents data in the Destination Address field, the **ts_in** represents the timestamp data in form of hexadecimal number, and the **key_in** represents the HMAC secret key.

In addition, the self-authentication options might be adapted to use with IPv6 (Internet Protocol version 6). The options format look like the format used on IP (version 4). But there are some differences about the option-type octet and the IPv6 header that is used for carrying the self-authentication options. For option-type octet, the format is changed but it still can be defined within 1 octet of option-type, by following the IPv6 standard [10] in part of Type-Length-Value (TLV) encoding. About IPv6 header, there is no IP options field in the header. Hence, self-authentication options will be carried by one of IPv6 extension headers called Hop-by-Hop options header. It carries optional information that must be examined by every node along a delivery path of IPv6 packet. The options header is identified by a Next Header value of 0 in the IPv6 header. The design of self-authentication options for IPv6 can be illustrated in figure 4.

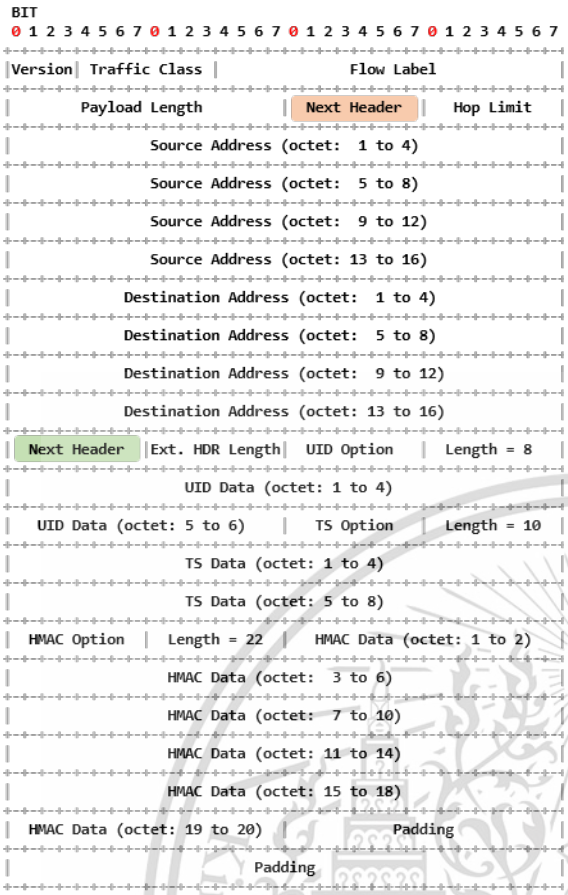


Figure 4 IPv6 header with self-authentication options

3.2. Data Flow and Operations

The Linux operating system is used in this enhancement. In order to control the flow of IP packet and manipulate the IP header, using a netfilter framework and a Scapy [11] program is the right solution. The netfilter framework is used to provide IP traffic control on the host. The **POSTROUTING** chain **mangle** table of the framework is mainly used to do mangling on IP packets. In the operation, the IP packets are redirected from kernel space to user space at this point (chain and table), by using the **NFQUEUE** (the Netfilter Queue). Scapy is a powerful interactive IP packet manipulation program written in Python. It works on the user space, and it can inject the IP packet back to the kernel space.

The self-authentication ability focuses on an outgoing IP traffic, an original IP packet is transmitted from client device in a local network through the NAC before reaching the destination. At the client device, the IP packet is redirected to the user space, then the IP header is modified by the IP header manipulation program, the modified IP packet is injected back to the kernel space, and is sent out to the network. The IP packet flow on the client device is shown in figure 5.

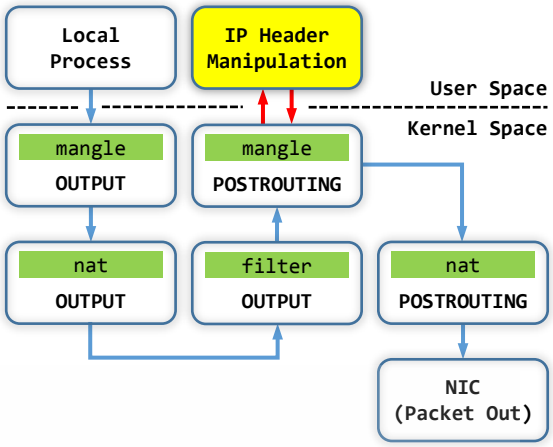


Figure 5 IP packet flow on the client device

At the NAC, the IP packet is received on an incoming network interface, then will be bridged to an outgoing network interface. The IP packet is redirected to the IP header manipulation program on user space for verifying the self-authentication options in the IP header. For any valid IP packet, the self-authentication options are deleted from the IP header, and the modified IP packet is injected back to the kernel space and is sent out to the network. The IP packet flow on the NAC is shown in figure 6.

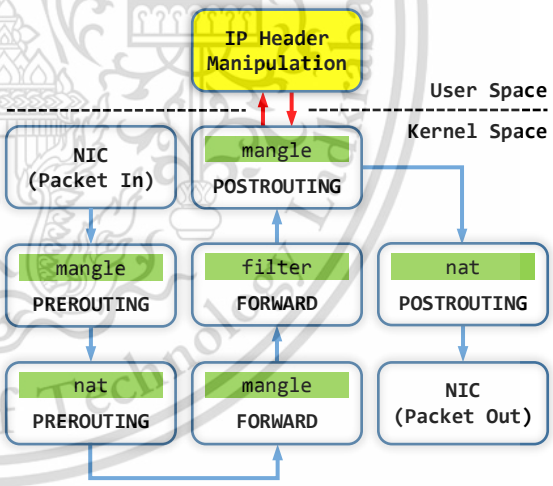


Figure 6 IP packet flow on the NAC

For the operation of IP header manipulation on client device, the particular program running on user space deletes all IP options (if any) in the incoming IP packet, then adds the UID option, the TS option, and the HMAC option respectively. After adding self-authentication options, the IHL value in the IP header is increased to 15 (1111 in binary), the Header Checksum value is recalculated as well. Finally, the modified IP packet is sent back to the kernel space.

An action on IP packet on the NAC consists of self-authentication verification and IP header manipulation. All actions are described in figure 7.

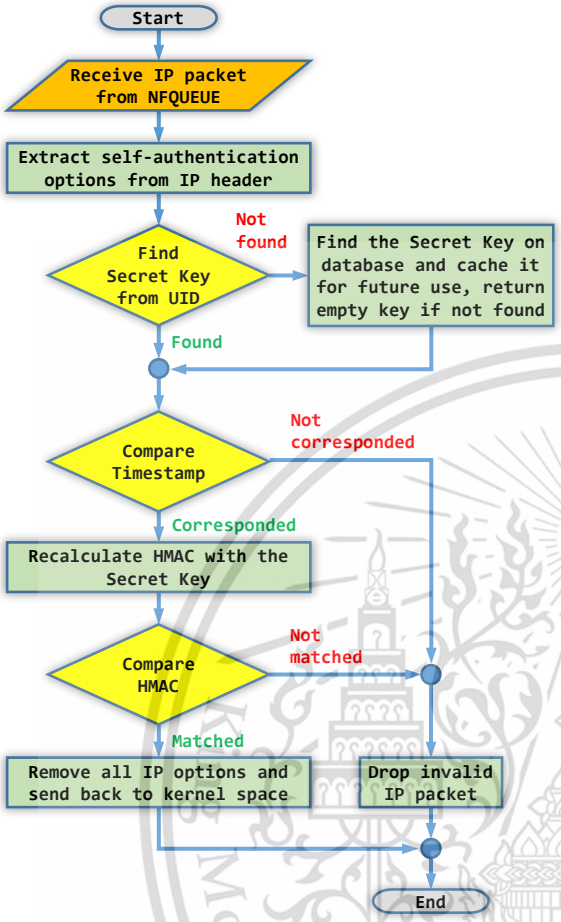


Figure 7 Self-authentication verification

In part of self-authentication verification on the NAC, self-authentication options are extracted from the incoming IP header. The particular program will find the secret key corresponding the UID option in the cache. The timestamp is firstly checked. The HMAC is recalculated with obtained secret key and is compared with the HMAC option. IP packets that do not pass timestamp checking and HMAC comparison are dropped.

4. Experimentation and Results

There are four computers, two network switches, and one Internet router in the experimental environment. The computers work as a client device, a NAC, and analyzers. The test IP packet is sent from the client device to the Internet. IP packet passing the network switches is copied and sent to the analyzers. IP packets captured on the analyzers are used as an experimental result. Figure 8 shows the experimental topology.

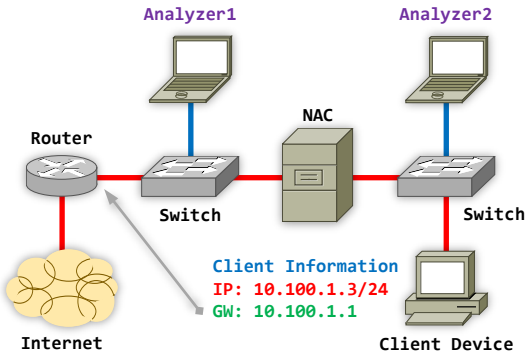


Figure 8 Experimental topology

In the figure, the client device has one NIC (Network Interface Card) named **ens160**, the NAC has two NICs named **ens160** (router side) and **ens192** (client side). The NAC operates in bridge mode, the two physical NICs are member of the bridge. In order to redirect IP packet to the IP header manipulation program, the following **iptables** commands are executed on both NAC and client device.

```
# # IPTables command for Client Device.
# iptables -t mangle -I POSTROUTING \
> -o ens160 -j NFQUEUE --queue-num 0

# # IPTables command for NAC.
# iptables -t mangle -I POSTROUTING \
> -m physdev --physdev-out ens160 \
> -j NFQUEUE --queue-num 0
```

After sending the test ICMP packet from the client device (10.100.1.3) to the destination (8.8.8.8), Captured IP packet on the analyzer1 and the analyzer2 is decoded as displayed in figure 9 and 10.

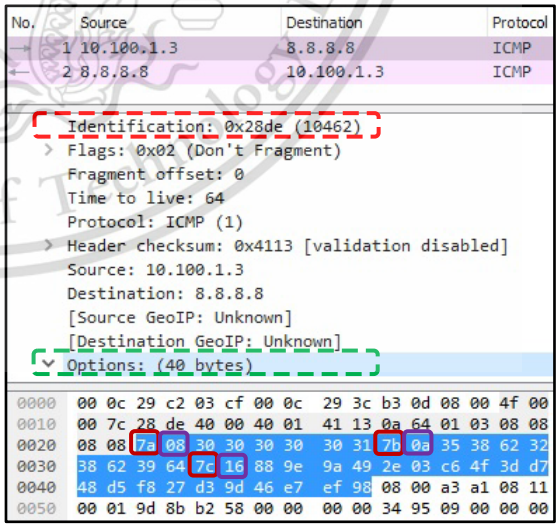


Figure 9 IP packet decoded on the analyzer1

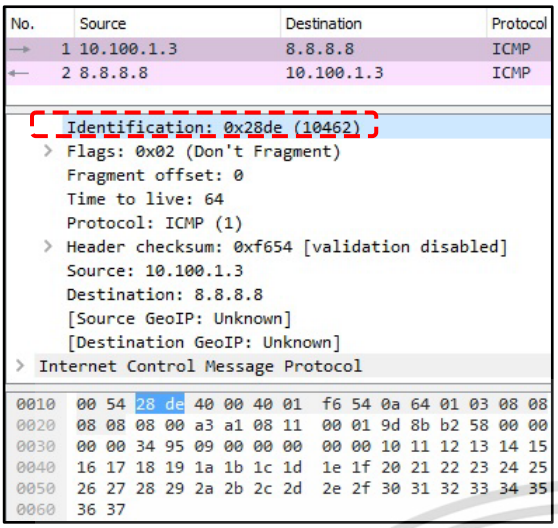


Figure 10 IP packet decoded on the analyzer2

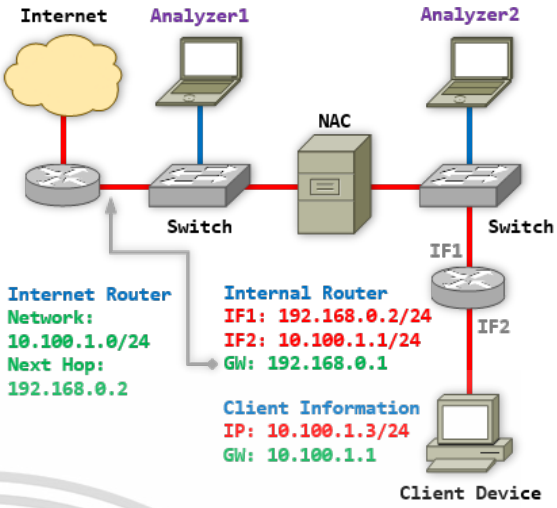


Figure 12 Experimental topology 2

In figure 9, the self-authentication options are attached to the test IP packet (ICMP) sent from the client device. The identification number of the IP header is 10462. The self-authentication options including UID option (0x7A), TS option (0x7B), and HMAC option (0x7C), as described in the section of implementation, are inserted into the IP options field. After passing the self-authentication verification, the self-authentication options are removed and sent on a wire as shown in figure 10. In another view on the NAC, the IP header manipulation program running in verbose mode also displays the same information of the test IP packet, which is shown in figure 11.

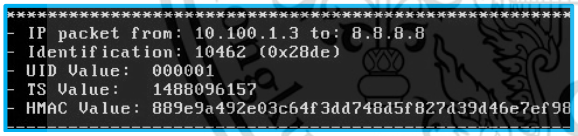


Figure 11 Test IP packet displayed on the NAC

In another one experimentation, the NAC is placed between routers. The test IP packet is sent from client device and is routed to an Internet router by an internal router. Network configuration and **iptables** commands used on both client device and NAC same as previous experimental topology. IP packet passing the network switches and the NAC, is copied and sent to the analyzers. IP packets captured on the analyzers are used as an experimental result. The experimental topology 2 is shown in figure 12.

About results of sending the test IP packet from client device, self-authentication options inserted in the IP header are decoded and displayed on the analyzer2, as shown in figure 13. After passing the verification process on NAC, the self-authentication options are removed. The same IP packet without any IP options is decoded and displayed on the analyzer1, as shown in figure 14. All operations and results like the previous experimentation. But there is one thing different, the TTL (Time-to-Live) of the test IP packet is decreased, from value of 64 to 63, because of the routing process.

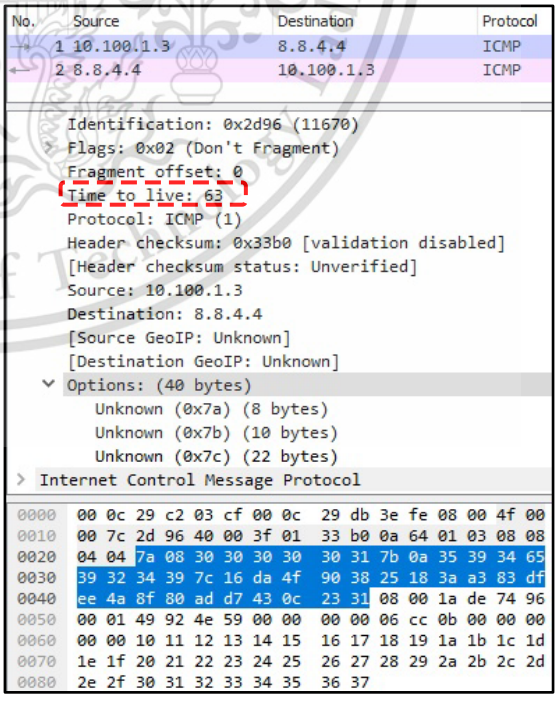


Figure 13 IP header with self-authentication options

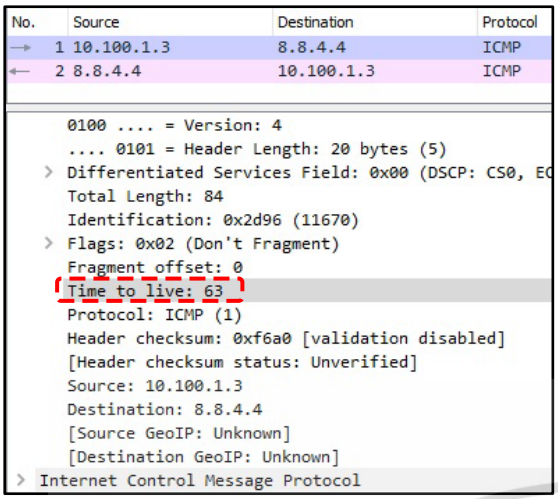


Figure 14 IP header after passing the NAC

In case of reliability test, the IP header manipulation program running on client device has been modified, to simulate sending of some IP packets through the NAC. The conditions and results of the simulation are list in table 1.

Item	Conditions	Accept	Drop
1	Sending packet with self-authentication options corresponding the design	•	
2	Timestamp option on client device is increased/decreased before calculating the HMAC		•
3	Using different secret key between NAC and client device for IP packet with same UID in self-authentication options		•
4	Input parameters for calculating the HMAC do not meet the specifications		•
5	Sending packet without attaching the self-authentication options in IP header		•

Table 1 Self-authentication verification test results

In addition, when considering about the mentioned network access control which are Captive Portal, IPsec AH, and self-authentication options, the key features of each methods can be compared as shown in table 2.

Features \ Method	IP options	IPsec AH	Captive Portal
Control accessing networks	Yes	Yes	Yes
Anti-spoofing	Yes	Yes	No
Per-packet self-authentication	Host-to-Any	Host-to-Host	No
Not require for authentication before data communication	Yes	No	No
Not require for session maintenance	Yes	No	No

Table 2 A comparison of access control applications

5. Conclusion

User authentication in an Internet Protocol provides a new mechanism for verifying an identity of device owner or computer user. It is especially suitable to use with network access control application. The key feature is a design of self-authentication options. Data communication at IP layer can be controlled by the NAC which can be placed everywhere on the communication path. It is similar to IPsec AH but it also works on LAN. Without needing additional user authentication process and there is no session to be maintained, it makes data communication more smooth and seamless. For security awareness, it also provides a mitigation of replay attack and a prevention of sending source-spoofed IP packet, which make identifying the user more reliability. The enhancement makes user authentication mechanism look different.

6. References

[1] C. Manusankar, S. Karthik, and T. Rajendran, "Intrusion Detection System with Packet Filtering for IP Spoofing," International Conference on Communication and Computational Intelligence, India, pp. 563-567, December, 2010.

[2] G. Appenzeller, M. Roussopoulos, and M. Baker, "User-Friendly Access Control for Public Network Ports," INFOCOM IEEE, vol. 2, pp. 699-707, March, 1999.

[3] S. Kent and K. Seo, "Security Architecture for the Internet Protocol," RFC 4301, December, 2005.

[4] S. Kent, "IP Authentication Header," RFC 4302, December, 2005.

[5] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104, February, 1997.

[6] D. Eastlake and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," RFC 3174, September, 2001.

[7] P. Syverson, "A taxonomy of replay attacks," IEEE Computer Society Press, pp. 187-191, 1994.

[8] D. Denning and G. Sacco, "Timestamps in Key Distribution Protocols," Communications of the ACM, Vol. 24, pp. 533-536, August, 1981.

[9] J. Postel, "INTERNET PROTOCOL," RFC 791, September, 1981.

[10] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460, December, 1998.

[11] P. Biondi, "Packet generation and network based attacks with Scapy," CanSecWest/core05, France, May, 2005.

Python Source Code

- Packet manipulation program for a client device

```
#!/usr/bin/python

import socket
import nfqueue
import hmac
import hashlib
import time

from scapy.layers.inet import *

user_hmac_type = 1
user_id_number = 56601420
user_secret_key = "P@ssw0rd"

def int_to_hexstring(integer_in):
  num_hex = hex(integer_in)
  num_hex_str = str(num_hex)[2:]
  num_hex_str_zfill = num_hex_str.zfill(8)
  return num_hex_str_zfill

def gen_icv(packet_in, unix_time_in):

  if user_hmac_type == 1:
    icv = hmac.new(user_secret_key, "", hashlib.md5)
  else:
    icv = hmac.new(user_secret_key, "", hashlib.sha1)

  icv.update(str(packet_in.version))
  icv.update(str(packet_in.ihl))
  icv.update(str(packet_in.len))
  icv.update(str(packet_in.id))
  icv.update(str(packet_in.proto))
  icv.update(str(packet_in.src))
  icv.update(str(packet_in.dst))
  icv.update(str(packet_in.payload))
  icv.update(str(unix_time_in))

  return icv.digest()

def nfqueue_callback(payload_in):
  data = payload_in.get_data()
  packet = IP(data)

  ## Displaying packet information ##

  print ""
  print "Packet SRC Address: "+packet[IP].src
  print "Packet DST Address: "+packet[IP].dst
  print "Packet HDR Length (before processing): "+str(packet[IP].ihl)+" [32-bit words]"
  print "Packet ALL Length (before processing): "+str(packet[IP].len)+" [bytes]"
  print "Packet Identification: "+str(packet[IP].id)

  ## Preparing authentication data ##

  user_id = int_to_hexstring(user_id_number)
  opt_id = user_id.decode("hex")
  unix_time = int(time.time())
  user_time = int_to_hexstring(unix_time)
  opt_time = user_time.decode("hex")

  print "User ID Number: "+str(user_id_number)+" [Integer], "+user_id+" [Fixed HEX]"
  print "User Unix Time: "+str(unix_time)+" [Integer], "+user_time+" [Fixed HEX]"

  ## Packet manipulation ##

  del packet[IP].options
  if (user_hmac_type == 1):
```

```
    packet[IP].ihl += 8
    packet[IP].len += 32
    opt_hmac = gen_icv(packet, unix_time)
    packet[IP].options = IPOption("%s%s%s%s"%("\x7a\x06"+opt_id, "\x7b\x06"+opt_time,
"\x7c\x12"+opt_hmac, "\x01\x00"))
    print "Packet AUTH Data: "+opt_hmac.encode("hex")+" [HMAC-MD5]"
  else:
    packet[IP].ihl += 9
    packet[IP].len += 36
    opt_hmac = gen_icv(packet, unix_time)
    packet[IP].options = IPOption("%s%s%s%s"%("\x7a\x06"+opt_id, "\x7b\x06"+opt_time,
"\x7c\x16"+opt_hmac, "\x01\x00"))
    print "Packet AUTH Data: "+opt_hmac.encode("hex")+" [HMAC-SHA1]"

  print "Packet HDR Length (after processing): "+str(packet[IP].ihl)+" [32-bit words]"
  print "Packet ALL Length (after processing): "+str(packet[IP].len)+" [bytes]"

  del packet[IP].chksum
  payload_in.set_verdict_modified(nfqueue.NF_ACCEPT, str(packet), len(packet))

def main():
  q = nfqueue.queue()
  q.open()
  q.bind(socket.AF_INET)
  q.set_callback(nfqueue_callback)
  q.create_queue(0)
  try:
    q.try_run()
  except KeyboardInterrupt:
    q.unbind(socket.AF_INET)
    q.close()

main()
```

Python Source Code

- Packet manipulation program for an authenticator

```
#!/usr/bin/python

import socket
import nfqueue
import hmac
import hashlib
import sqlite3

from scapy.layers.inet import *

time_diff = 1
secret_keys = {}

def gen_icv(packet_in, unix_time_in, hmac_len_in, hmac_key_in):

  if (hmac_len_in == 18):
    icv = hmac.new(hmac_key_in, "", hashlib.md5)
  else:
    icv = hmac.new(hmac_key_in, "", hashlib.sha1)

  icv.update(str(packet_in.version))
  icv.update(str(packet_in.ihl))
  icv.update(str(packet_in.len))
  icv.update(str(packet_in.id))
  icv.update(str(packet_in.proto))
  icv.update(str(packet_in.src))
  icv.update(str(packet_in.dst))
  icv.update(str(packet_in.payload))
  icv.update(str(unix_time_in))

  return icv.digest()

def get_key_external(user_id_int_in):
  conn = sqlite3.connect("secret_keys.db")
  c = conn.cursor()
  c.execute("select db_key from skeys where db_id=:uid", {"uid":user_id_int_in})
  row = c.fetchone()
  if (row is None):
    conn.close()
    return None
  else:
    conn.close()
    return str(row[0])

def nfqueue_callback(payload):
  data = payload.get_data()
  packet = IP(data)

  ## Displaying packet information ##

  print ""
  print "Packet SRC Address: "+packet[IP].src
  print "Packet DST Address: "+packet[IP].dst
  print "Packet HDR Length (before processing): "+str(packet[IP].ihl)+" [32-bit words]"
  print "Packet ALL Length (before processing): "+str(packet[IP].len)+" [bytes]"
  print "Packet Identification: "+str(packet[IP].id)

  ## Checking and gathering valid authentication options ##

  options_count = len(packet[IP].options)

  if (options_count < 3):
    print "Authentication: DROP (the amount of required IPv4 options is not enough)"
    payload.set_verdict(nfqueue.NF_DROP)
    return

  check = 0
```

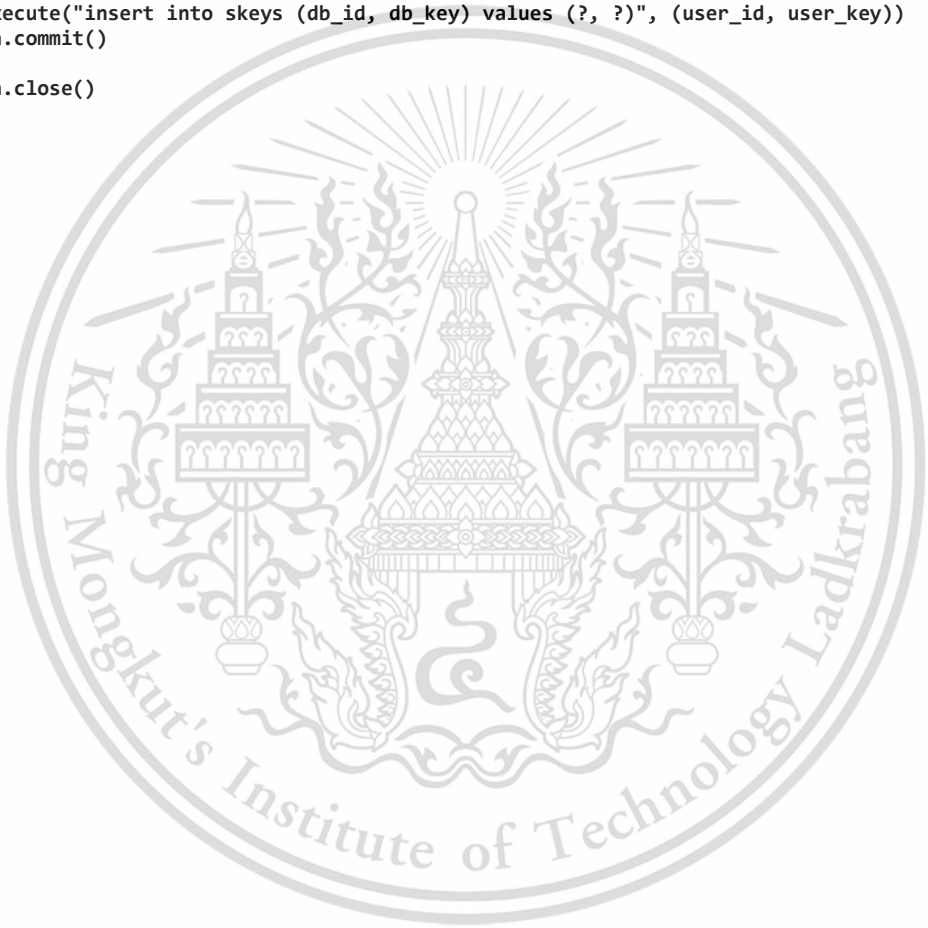
```
opt_num_sum = 0
user_id = 0
user_time = 0
user_hmac = ""
user_hmac_len = 0

for i in range(0, options_count):
  if (packet[IP].options[i].option == 26):
    check += 1
    opt_num_sum += packet[IP].options[i].option
    #print packet[IP].options[i].value.encode("hex")
    #print int(packet[IP].options[i].value.encode("hex"), 16)
    user_id = int(packet[IP].options[i].value.encode("hex"), 16)
  if (packet[IP].options[i].option == 27):
    check += 1
    opt_num_sum += packet[IP].options[i].option
    #print packet[IP].options[i].value.encode("hex")
    #print int(packet[IP].options[i].value.encode("hex"), 16)
    user_time = int(packet[IP].options[i].value.encode("hex"), 16)
  if (packet[IP].options[i].option == 28):
    check += 1
    opt_num_sum += packet[IP].options[i].option
    #print packet[IP].options[i].value.encode("hex")
    user_hmac = packet[IP].options[i].value.encode("hex")
    user_hmac_len = packet[IP].options[i].length

if ((check != 3) or (opt_num_sum != 81)):
  print "Authentication: DROP (there is IPv4 option using an invalid option number)"
  payload.set_verdict(nfqueue.NF_DROP)
  return

## Verifying authentication options ##

auth_unix_time = int(time.time())
print "-- IPv4 Option: Timestamp --"
print "Unix Time on Client: "+str(user_time)
print "Unix Time on Authenticator: "+str(auth_unix_time)
if (abs(auth_unix_time - user_time) > time_diff):
  print "Authentication: DROP (different time is more than acceptable value)"
  payload.set_verdict(nfqueue.NF_DROP)
  return

user_id_string = str(user_id)
print "-- IPv4 Option: User Identification --"
print "User Identification: "+user_id_string
if (user_id_string in secret_keys):
  user_key = secret_keys[user_id_string]
  print "Key in dictionary is used for HMAC computation"
else:
  print "Requesting key from external database"
  user_key = get_key_external(user_id)
  if (user_key is None):
    print "Authentication: DROP (there is no key associated with this user)"
    payload.set_verdict(nfqueue.NF_DROP)
    return
  else:
    secret_keys[user_id_string] = user_key
    print "Found, key is added into dictionary"

print "-- IPv4 Option: Packet Authentication --"
if (user_hmac_len == 18):
  print "Client HMAC Length: "+str(user_hmac_len-2)+" [bytes] [HMAC-MD5]"
  auth_hmac = gen_icv(packet, user_time, user_hmac_len, user_key).encode("hex")
else:
  print "Client HMAC Length: "+str(user_hmac_len-2)+" [bytes] [HMAC-SHA1]"
  auth_hmac = gen_icv(packet, user_time, user_hmac_len, user_key).encode("hex")

print "HMAC Value: "+auth_hmac+" [Authenticator]"
print "HMAC Value: "+user_hmac+" [Client]"
if (auth_hmac != user_hmac):
  print "Authentication: DROP (packet authentication fails)"
  payload.set_verdict(nfqueue.NF_DROP)
  return
else:
```

```
    print "Authentication: ACCEPT"

  ## Removing all IPv4 options ##

  del packet[IP].options
  packet[IP].ihl = 5
  packet[IP].len = len(packet)
  del packet[IP].chksum

  print "Packet HDR Length (after processing): "+str(packet[IP].ihl)+" [32-bit words]"
  print "Packet ALL Length (after processing): "+str(packet[IP].len)+" [bytes]"

  payload.set_verdict_modified(nfqueue.NF_ACCEPT, str(packet), len(packet))

def main():
  q = nfqueue.queue()
  q.open()
  q.bind(socket.AF_INET)
  q.set_callback(nfqueue_callback)
  q.create_queue(0)
  try:
    q.try_run()
  except KeyboardInterrupt:
    q.unbind(socket.AF_INET)
    q.close()

main()
```

Python Source Code

- User management on an external database

```
#!/usr/bin/python

import sqlite3

conn = sqlite3.connect("secret_keys.db")

#conn.execute("create table skeys (db_id int primary key not null, db_key text not null)")

user_id = 56601420
user_key = "P@ssw0rd"

c = conn.cursor()
c.execute("insert into skeys (db_id, db_key) values (?, ?)", (user_id, user_key))
conn.commit()

conn.close()
```



AUTHOR BIOGRAPHY

Name	Mr. Parinya Thamthawornsakul
Date of Birth	28 February 1983, in Udon Thani
Address	988 Moo 12, Sawang Daen Din Sub-district
	Sawang Daen Din District, Sakon Nakhon Province
	47110
Education	2008, Bachelor of Engineering, Telecommunication Engineering
	King Mongkut's Institute of Technology Ladkrabang
Specialization	1) Computer Network
	2) Virtualization and Cloud Computing Technology
Position and Place of Work	
2008 - 2010	Engineer
	Fastcomm Co., Ltd.
2010 - 2012	Programming Instructor
	Triam Udom Suksa School of the Northeast
2012 - 2015	Engineer
	Telecom One (Thailand) Limited
2015 - Present	Engineer
	Internet Data Center Department, CAT Telecom Public Co., Ltd.