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.

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

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.

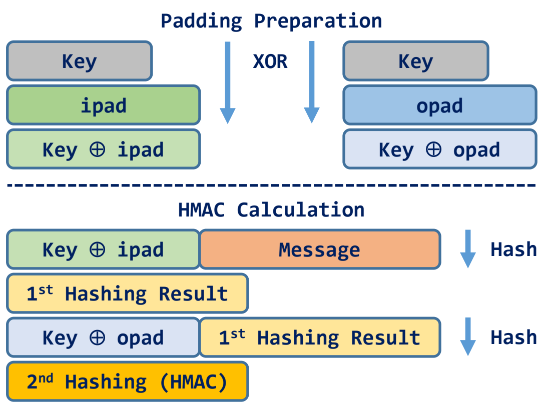


Figure 1 HMAC calculation

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.

```
BIT
0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Option-Type   | Option-Length |        Option-Data (variable)  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

+-+-+-+-+-+-+-+-+       1 is Copied Flag    (use 1 bit)
|1| 2 |    3    |       2 is Option Class   (use 2 bits)
+-+-+-+-+-+-+-+-+       3 is Option Number  (use 5 bits)
```

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.

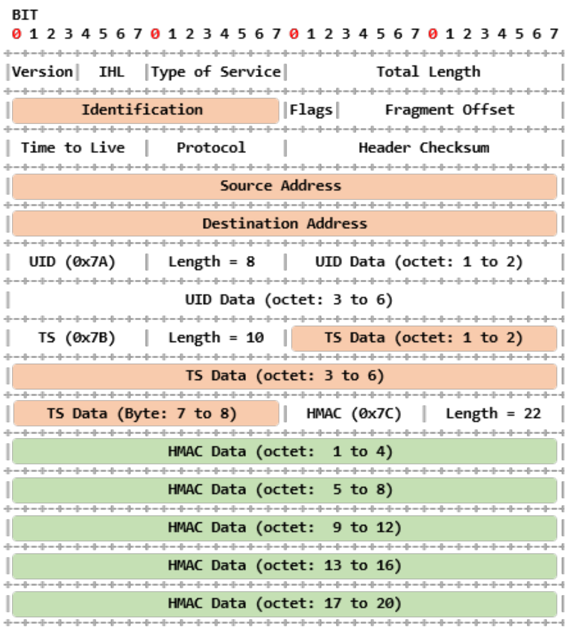


Figure 3 IP header with self-authentication options

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.

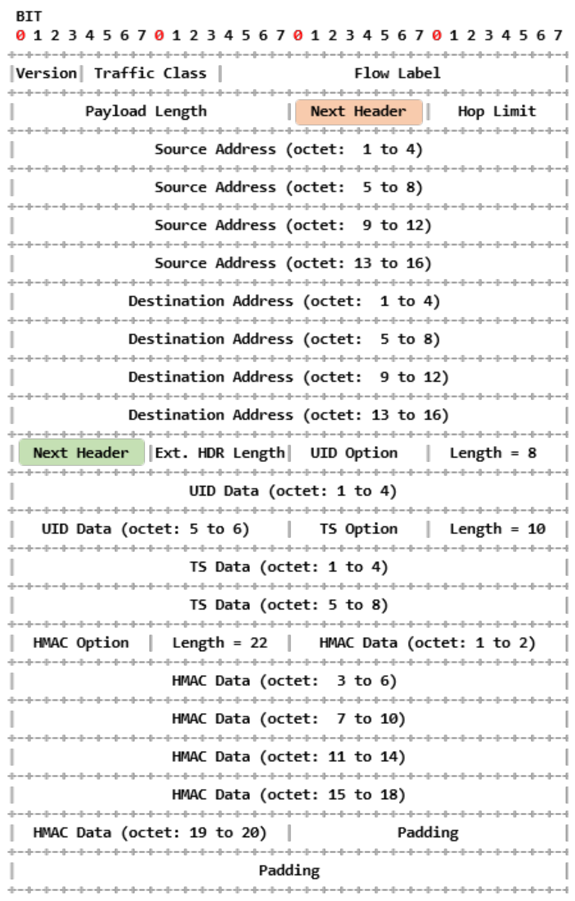

BIT
0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
Version
Traffic Class
Flow Label
Payload Length
Next Header
Hop Limit
Source Address (octet: 1 to 4)
Source Address (octet: 5 to 8)
Source Address (octet: 9 to 12)
Source Address (octet: 13 to 16)
Destination Address (octet: 1 to 4)
Destination Address (octet: 5 to 8)
Destination Address (octet: 9 to 12)
Destination Address (octet: 13 to 16)
Next Header
Ext. HDR Length
UID Option
Length = 8
UID Data (octet: 1 to 4)
UID Data (octet: 5 to 6)
TS Option
Length = 10
TS Data (octet: 1 to 4)
TS Data (octet: 5 to 8)
HMAC Option
Length = 22
HMAC Data (octet: 1 to 2)
HMAC Data (octet: 3 to 6)
HMAC Data (octet: 7 to 10)
HMAC Data (octet: 11 to 14)
HMAC Data (octet: 15 to 18)
HMAC Data (octet: 19 to 20)
Padding
Padding

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.

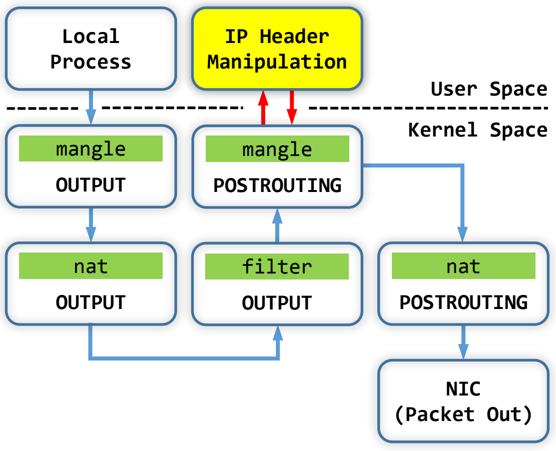


Figure 5 IP packet flow on the client device

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.

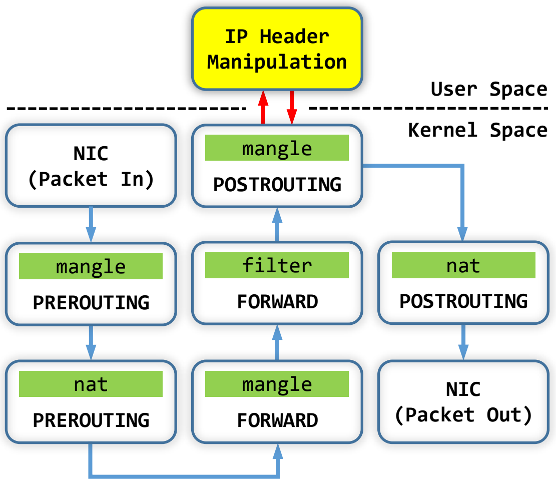


Figure 6 IP packet flow on the NAC

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.

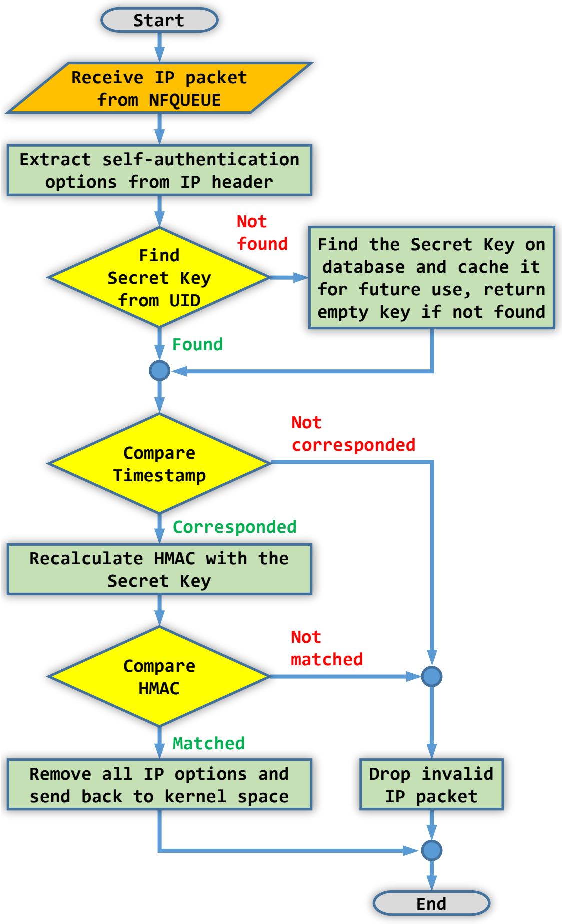

Start
Receive IP packet from NFQUEUE
Extract self-authentication options from IP header
Find Secret Key from UID
Not found
Find the Secret Key on database and cache it for future use, return empty key if not found
Found
Compare Timestamp
Not corresponded
Corresponded
Recalculate HMAC with the Secret Key
Compare HMAC
Not matched
Matched
Remove all IP options and send back to kernel space
Drop invalid IP packet
End

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.

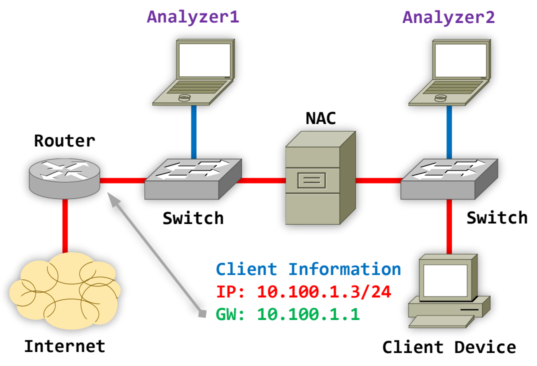


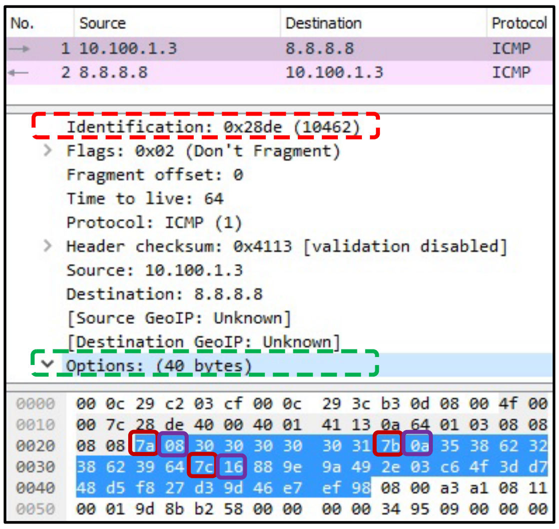
Figure 8 Experimental topology

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.

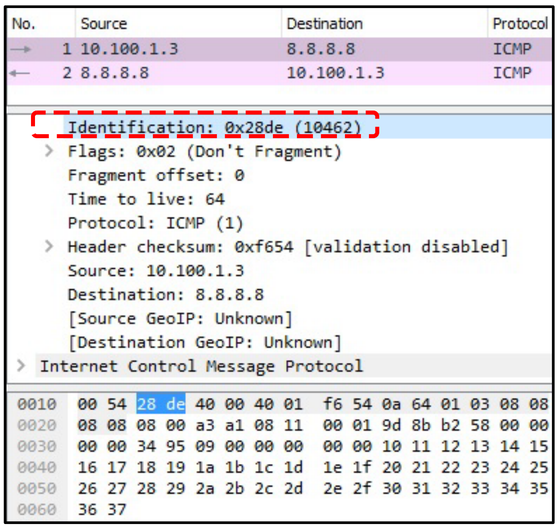


```
No.   Source            Destination       Protocol
→   1 10.100.1.3        8.8.8.8           ICMP
←   2 8.8.8.8           10.100.1.3        ICMP

    Identification: 0x28de (10462)
  > Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
  > Header checksum: 0x4113 [validation disabled]
    Source: 10.100.1.3
    Destination: 8.8.8.8
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
  ˅ Options: (40 bytes)

0000  00 0c 29 c2 03 cf 00 0c  29 3c b3 0d 08 00 4f 00
0010  00 7c 28 de 40 00 40 01  41 13 0a 64 01 03 08 08
0020  08 08 7a 08 30 30 30 30  30 31 7b 0a 35 38 62 32
0030  38 62 39 64 7c 16 88 9e  9a 49 2e 03 c6 4f 3d d7
0040  48 d5 f8 27 d3 9d 46 e7  ef 98 08 00 a3 a1 08 11
0050  00 01 9d 8b b2 58 00 00  00 00 34 95 09 00 00 00
```

Figure 9 IP packet decoded on the analyzer1



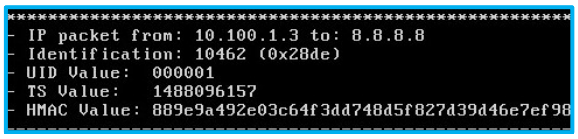
```
No.   Source          Destination      Protocol
→   1 10.100.1.3      8.8.8.8          ICMP
←   2 8.8.8.8         10.100.1.3       ICMP

    Identification: 0x28de (10462)
  > Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
  > Header checksum: 0xf654 [validation disabled]
    Source: 10.100.1.3
    Destination: 8.8.8.8
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
> Internet Control Message Protocol

0010  00 54 28 de 40 00 40 01  f6 54 0a 64 01 03 08 08
0020  08 08 08 00 a3 a1 08 11  00 01 9d 8b b2 58 00 00
0030  00 00 34 95 09 00 00 00  00 00 10 11 12 13 14 15
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35
0060  36 37
```

Figure 10 IP packet decoded on the analyzer2

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.



```
****************************************************************
- IP packet from: 10.100.1.3 to: 8.8.8.8
- Identification: 10462 (0x28de)
- UID Value:  000001
- TS Value:   1488096157
- HMAC Value: 889e9a492e03c64f3dd748d5f827d39d46e7ef98
----------------------------------------------------------------
```

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.

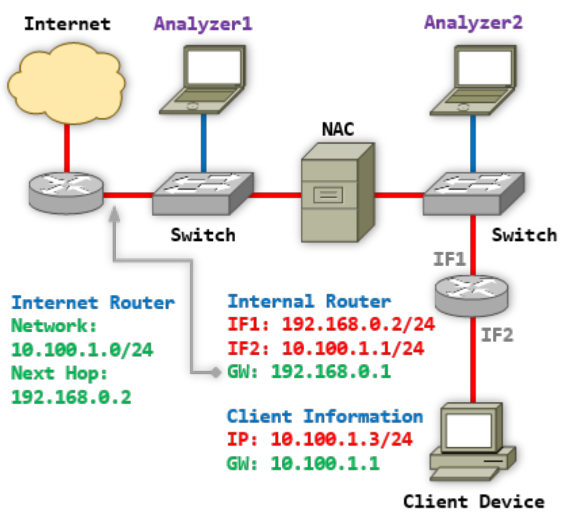
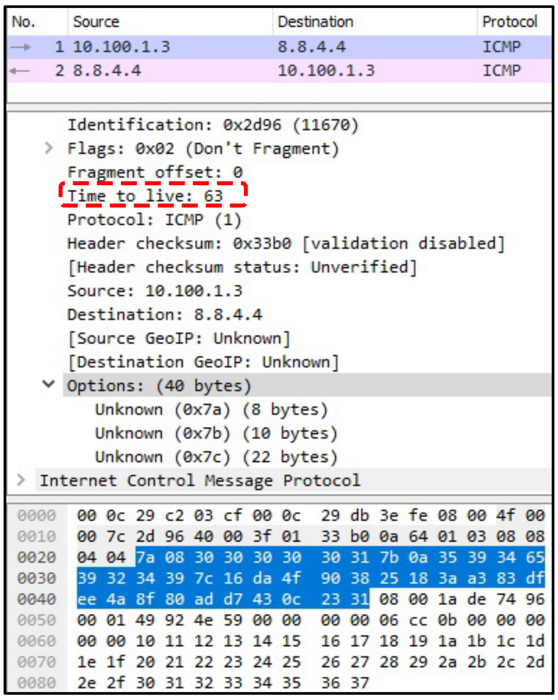


Figure 12 Experimental topology 2

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.

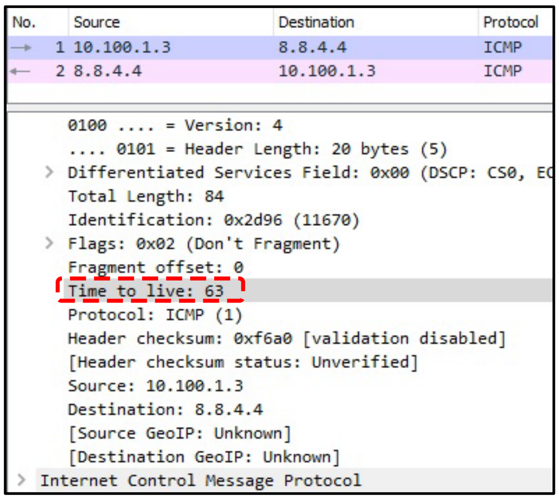


```
No.   Source          Destination      Protocol
→   1 10.100.1.3      8.8.4.4          ICMP
←   2 8.8.4.4         10.100.1.3       ICMP

    Identification: 0x2d96 (11670)
  > Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 63
    Protocol: ICMP (1)
    Header checksum: 0x33b0 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.100.1.3
    Destination: 8.8.4.4
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
  v Options: (40 bytes)
      Unknown (0x7a) (8 bytes)
      Unknown (0x7b) (10 bytes)
      Unknown (0x7c) (22 bytes)
> Internet Control Message Protocol

0000  00 0c 29 c2 03 cf 00 0c  29 db 3e fe 08 00 4f 00
0010  00 7c 2d 96 40 00 3f 01  33 b0 0a 64 01 03 08 08
0020  04 04 7a 08 30 30 30 30  30 31 7b 0a 35 39 34 65
0030  39 32 34 39 7c 16 da 4f  90 38 25 18 3a a3 83 df
0040  ee 4a 8f 80 ad d7 43 0c  23 31 08 00 1a de 74 96
0050  00 01 49 92 4e 59 00 00  00 00 06 cc 0b 00 00 00
0060  00 00 10 11 12 13 14 15  16 17 18 19 1a 1b 1c 1d
0070  1e 1f 20 21 22 23 24 25  26 27 28 29 2a 2b 2c 2d
0080  2e 2f 30 31 32 33 34 35  36 37
```

Figure 13 IP header with self-authentication options



```
No.  Source        Destination    Protocol
→  1 10.100.1.3    8.8.4.4        ICMP
←  2 8.8.4.4       10.100.1.3     ICMP

      0100 .... = Version: 4
      .... 0101 = Header Length: 20 bytes (5)
    > Differentiated Services Field: 0x00 (DSCP: CS0, EC
      Total Length: 84
      Identification: 0x2d96 (11670)
    > Flags: 0x02 (Don't Fragment)
      Fragment offset: 0
      Time to live: 63
      Protocol: ICMP (1)
      Header checksum: 0xf6a0 [validation disabled]
      [Header checksum status: Unverified]
      Source: 10.100.1.3
      Destination: 8.8.4.4
      [Source GeoIP: Unknown]
      [Destination GeoIP: Unknown]
  > Internet Control Message Protocol
```

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.

Method / Features	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.