Design and Implementation of Secure Tapping Alert Protocol

Gookwhan Ahn and Kwangjo Kim

IRIS(International Research center for Information Security),
ICU(Information and Communications University),
58-4, Hwaam-Dong, Yusung-Ku, Taejon, 305-732, Korea
{tachyon,kkj}@icu.ac.kr

Abstract. This paper proposes an improved protocol of original TAP[1], which detects if any computer is eavesdropping on the network. When tapping is detected, TAP alerts the network users on the danger of tapping. The improved TAP can additionally detect tapping on a remote computer of the remote network across routers.

An underlying model is revised for an attacker who sniffs the network and breaks down the improved protocol, and the requirements on the protocol in terms of security and functionality are defined. We also describe how to detect if a network interface card on a remote computer is in the sniffing mode. The proposed protocol employs cryptographic primitives to guarantee not only authentication of the code which monitors tapping operation, but also integrity and confidentiality of the data being sent.

This protocol should also use tamper-proof device to securely store the private key and session key. To achieve this, the protocol can employ a secure token such as smart card, iButton, etc. Finally, we verified that the proposed protocol can protect authorized users from illegal eavesdropping on the network.

1 Introduction

With the wide spread use of the Internet, numerous techniques have been devised such that attackers can break into the network, and then steal, forge, and destroy data in the target computer system. Tapping is one of the most popular network attacking methods appeared since early 1980s. In this paper we propose an improved protocol of TAP[1], which protects the system from tapping. Upon detection, it alerts the network users on the danger of tapping that can leak the information of the users. Tapping called by several different terminologies - snooping, sniffing or eavesdropping etc, in Local Area Network(LAN) is relatively easy since Ethernet is a shared network and all the packets on the network are broadcast in a network. When someone eavesdrops on a session of a connection while data are transferred between the computers, the person can easily extract the data. In addition, if the user-ids and its corresponding passwords are extracted through tapping, the eavesdropper can even break into the computer system. Due to this, tapping has been one of the most popular and easiest ways to penetrate a computer system. A number of sniffing programs on the Internet have been developed, and many attackers have used them for breaking into computer systems and networks. Tapping can be done by taking advantage of the promiscuous mode of network interface. Promiscuous mode is a condition when the Network Interface Controller (NIC) of a computer system passes all frames up to the higher network layers regardless of the destination address. Normally, a network controller passes up only the frames having the destination address to that device. However, when it is in promiscuous mode, the destination address is not checked. In other words, if an NIC is set to the promiscuous mode, it means that the computer is running the sniffing or tapping program [4]. Fortunately, there are some countermeasures developed to avoid tapping as follows[4]:

- Do not use the NIC that supports promiscuous mode.
- Use one-time password for connection.
- Encrypt a user's connection by using SSH (secure shell) program, etc.
- Use a switching hub so that an attacker cannot tap the network easily.
- Use tools such as "CPM" to detect the promiscuous mode of NIC on a local computer [15, 16].

In fact, since the tapping program can work in fully passive mode on the network, it is difficult (sometimes impossible) for other computers to detect if the programs are running on LAN. Nonetheless, there are also some methods to detect the tapping on a remote computer within LAN as follows[2]:

- OS kernel test : To exploit a weakness in the TCP/IP stack implementation of some OS (i.e. Linux, Windows 95/98/NT)
- DNS query test : To observe that a computer performs DNS queries to translate IP addresses of the captured packets.
- Network and machine latency tests : To create a large amount of network traffic for a short period of time.

In most cases, however, the sniffing program can be evasive about these methods, and they can not even detect the sniffing program where the computer lies on remote network across routers. For this reason, it is necessary to find out a specific method to discover the status of NIC on even remote network, not being evaded by the sniffing attack.

For this reason, we propose an improved method of original TAP[1] for detecting the promiscuous mode even on remote network. The new method, called also TAP-v2, uses cryptographic schemes [5, 6] in order to provide authentication of checking module and integrity of data. To hide sensitive data like keys for encryption and decryption, secure token is also used.

This paper is organized as follows: Section 2 discusses the fundamental issues related to tapping. In this section, we summarize the previous methods for detecting the tapping program. The security requirements and message formats for the proposed protocol are described in Section 3. In Section 4, the proposed protocol is presented, which prevents the computer systems and users from being tapped by any attacker, and the detailed description of the protocol is given. We explain how TAP-v2 is implemented, and analyze the protocol with respect to security and performance, and compare with the known detecting tools in Section 5. Finally, conclusions are given in Section 6.

2 Methods of Detection

This section reviews some techniques to detect when tapping is running on the computer and network. Some tools are known about how to detect the tapping on the local computer and on the remote computer in a network segment. However, as our motivation, we also describe that if these detecting methods are used, there are some ways that the tapping program can evade as described.

2.1 Detecting tapping on the local computer

If the OS of a computer is Unix, an attacker who has once gained root privilege can run a tapping program in order to obtain useful information from your network (for example, to retrieve passwords to other hosts on the network). In general, it is difficult to detect a process of sniffing, because the process name can be disguised as something normal. It can even be a Trojan horse program. The only way to detect the tapping program in this case is to check if the NIC is in promiscuous mode.

Here we present how to detect if an NIC under Unix is in promiscuous mode. The following C code describes the routine for NIC running in SunOS 4.x and Linux[15]. If the value of the if-statement surrounded by a box is true, then the NIC is in promiscuous mode.

```
int fd, itf;
struct ifreq buf[MAXINTERFACES];
⋮
if (!(ioctl(fd, SIOCGIFFLAGS, (char *) &buf[itf]))){
         if ( buf[itf].ifr_flags & IFF_PROMISC) {
             /* This NIC is in promiscuous mode */
      }
}
```

By using above subroutine, one application program of checking that NIC is in promiscuous mode is CPM (Check Promiscuous Mode).

Another method is to use a following command: *'ifconfig -a'*. All Unix systems provide a command *'ifconfig'* to offer similar characteristics above. This command displays the available NICs, and shows all the configuration information about them. The word PROMISC means that the NIC is in promiscuous mode. Running the command: *'ifconfig -a | grep PROMISC'* will be non-empty if one of the NICs is in promiscuous mode.

2.2 Detecting tapping on the remote computer

In practice, a network sniffer (tapping program) is totally passive and an interesting thing in TCP/IP stack of NT and Unix, makes most of workstations behave differently when a tapping program is running. Due to this, it is possible to detect the tapping even though the tapping program is running on the remote computer not the local computer. The followings explain how the tapping program is detected on the remote computer [2].

OS Kernel Tests(Linux,Windows95/98/NT). Under normal situations the NIC filters and discards packets that are not addressed to the machine MAC address or the broadcast Ethernet address. If the packet is destined to the machines, actual Ethernet address or broadcast Ethernet address are copied and passed up to the kernel for processing. When a NIC is placed in promiscuous mode, every packet is passed on to the OS to analyze and/or process.

This weakness in the implementation of TCP/IP stack of some Linux kernels is exploited to discover Linux-based sniffers. The data inside the Ethernet frame would consist of an IP packet with the correct IP address of the computer or the broadcast address of the network. Various Linux kernels could see only the IP address in the packets, in order to determine whether the packets should be handed to the kernel stack for processing inside the local system. To exploit this, the tool creates a packet with MAC address that is not destined to any particular NIC, but contains a valid IP packet with the correct IP address of destination hosts. Vulnerable Linux kernels with the NIC in promiscuous mode examine the IP address and pass the packet to the appropriate stack. By creating an ICMP echo request containing the fake Ethernet frame, the vulnerable systems respond when the NIC is in promiscuous mode, but correctly ignore the packet when the NIC is not in this monitoring state. Thus, it means the status of the machine is exposed.

Various NetBSD kernels and Windows95/98/NT have exhibited similar characteristics to the above Linux.

DNS query tests. The DNS query tests operate on the premise that many of network data gathering tools perform IP address to name inverse resolution to provide DNS names in place of IP addresses. Therefore, the tapping program will perform DNS queries to translate IP addresses of the captured packets. To check for this, fake packets are transmitted, containing a fake IP address as the destination address of the packet, and being watched for DNS queries to the IP address. The computers that perform a DNS query to the IP address, are watching the packets that are not sent to themselves, and therefore are probably in promiscuous mode. This comes from the fact that the computers do not need to perform a DNS query for fake packets when NIC is not in promiscuous mode. Due to this, the status of NIC is also observed on the remote computer.

Network and Machine Latency Tests. The computers that monitor all incoming traffic must process all the information on the network, causing a heavy load on the computer. One of detecting methods of the tapping program on the remote computer can be done by measuring the average response time of the computer, then flooding the network with useless traffic, and measuring the response time again. A computer that monitors all network traffic, must be busier than the other computers, therefore the computer takes time to respond. To obtain more accurate result, this procedure should be performed a few times repeatedly, using different measuring methods. This is the most powerful method to spot computers on the LAN that are in promiscuous mode regardless of their OS. The important point to which that we must pay attention is that this method has the potentiality to create a large amount of network traffic for short periods of time.

Evading the detecting of tapping. There are some applications that detect the tapping program on the remote computer by using these characteristics just mentioned before. They are called *Antisniff* and *Neped*.

These tools work against most common sniffing tools because of using mentioned characteristics, but these tools are not perfect. Some methods are already suggested to evade the detection of tapping program on the remote computer, to hide the tapping programs, and to make them undetectable. Normally, the attacks are from legitimate computers on the network, and usually using standard sniffing tools. But even those standard tools can be hidden from the methods to detect the tapping program as follows[3]:

- Use a modified kernel or IP stack that does not suffer from the mentioned characteristics (Linux kernel 2.2.10 works correctly and drops incoming packets not destined for this Ethernet address.) Then, the tapping program running on this kernel will not be detected by *OS Kernel Tests*.
- Don't perform DNS query (most of tapping programs have this feature). This will pass *DNS query Tests*.
- Stop the tapping program when the network traffic exceeds a certain rate, passing *Network and Machine Latency Tests*.

As mentioned above, there exist some methods to evade the detecting methods on a remote computer in a network segment. This is our motivation. In Section 3, we will propose the new protocol to solve these problem. That is to say, it is to be capable of detecting the tapping program on a remote computer on LAN as well as on a remote network, and not being evaded by the tapping program.

3 Security Requirements and Message Formats

In this section, possible attacks and security requirements against this new protocol are presented. Also, message formats for the protocol are defined, and the protocol description explains how it works.

3.1 Requirements

Here we do not consider how to prevent the root privilege from being compromised. There are many techniques protecting a computer from the attack on the root privilege through Internet. Under this assumption, we consider the possible attacks and the requirements on the protocol in terms of security and functionality. This protocol basically requires 2 properties:

- To detect the tapping program on remote network across routers.
- No method to be evaded by the tapping program.

The following describes the mentioned properties in rather detail.

Possible Attacks. It is assumed that the attacker obtains the root privilege in order to eavesdrop on the network or stop the protocol inside computer systems since there is no tapping method without the root privilege.

When a protocol handling tapping is installed on a network, an attacker may tackle it in the following ways:

1. *Routine interruptability*: The attacker stops the routine running on memory. If an attacker wants not to send the result when the NIC is in promiscuous or not to the request computer or network, the routine will stop while running.
2. *Routine forgeability*: The attacker forges the routine checking the NIC. If an attacker wants to deceive the network users and computer system, it replaces the original checking routine with a forged one. The forged one conducts as the original one does, and the network users and computers will misjudge it to be a legitimate one.
3. *Data forgeability*: The attacker forges the data transmitted to other computers. Here if an attacker wants to provide the network with the routine checking the NIC and wants it to run normally but still wants to deceive the network users and computers, it forges the data being sent.

Security requirements.

- *Authentication*: If legitimate routines are needed to run normally and they are not to be forged, authentication is necessary.
- *Integrity*: If the data transferred through the network are not to be forged, integrity is necessary.
- *Confidentiality*: If the session key between two entities and private key for each entity are stored in a secure place, confidentiality is also necessary.

Functionality requirements.

- *Remote detect*: This protocol must provide the method to detect the promiscuous mode on remote network across routers. Therefore, if TAP-v2 is used in the Internet, everyone can check out the NIC of target computer to discover the sniffing mode.

3.2 Notations and Message Formats

Notations. We employ the following notations to represent the messages used in the proposed protocol.

$\|\|$	Concatenation of messages.
I	Initiator, who sends a message.
R	Responder, who responds to a message.
A $\Rightarrow$ B : M	Entity A sends a message M to entity B.
$DEC_X()$	Decrypt data with the private key of entity X.
$ENC_X()$	Encrypt data with the public key of entity X.
$K = g^{xy}$	Session key by Diffie-Hellman Key agreement.
$h_K()$	Keyed hash function.
$IPaddr_X$	IP address of entity X.
$MachineTypes_X$	A kind of OS and CPU (Linux, Solaris, AIX, etc.) of X.
PK_X	Public key of entity X.
SK_X	Secret key of entity X.
$Sign_X()$	Signature data with the private key of X.
Routine()	The program checking the promiscuous mode of an NIC on the main memory.
$TimeStamp_X$	TimeStamp of entity X.
'Y' or 'N'	'Y' means that the computer is in sniffing mode, but 'N' means it is not.

Message Formats. We design a set of message formats as follows:

- msg_n : Messages transmitted among the entities.

  msg_1=(h_K(Routine($TimeStamp_I$)) $\|$ Num of Tap. $\|$ $MachineTypes_R$
  msg_2= Num of Tap. $\|$ $TimeStamp_R$ $\|$ $MachineTypes_I$

- **TAPinit**: This message is first sent to R to build a session key using the Diffie-Hellman key exchange setting between two entities, I and R.

  ('TAPinit' $\|$ g^x)

- **TAPack**: This is used for precluding man-in-the-middle attack against building the session key as a shared key.

$$(\text{`TAPack'} \parallel g^y \parallel g^x \parallel h_k(g^y \parallel g^x\ )\ )$$

- **TAPagree**: This is used as the final step to setup a session key. As this message is sent to R which is one of the two entities on the key agreement protocol, it is said that two entities share a session key without any attack.

$$(\text{`TAPagree'} \parallel h_k(g^x \parallel g^y)\ )$$

- **TAPreq**: Request message to check the status of Responder, which may be running a sniffing program.

$$(\text{`TAPreq'} \parallel ENC_K(TimeStamp_I)\ )$$

- **TAPreply**: Response message of the checking routine to **TAPreq**.

$$(\text{`TAPreply'} \parallel ENC_K(msg_1)\ )$$

- **TAPalert**: Advertisement message for alerting the hosts on the danger of sniffing. This message is advertised by using the broadcast IP address on LAN.

$$(\text{`TAPalert'} \parallel Sign_I(msg_2) \parallel msg_2\ )$$

4 Proposed Protocol

This section proposes a new protocol called TAP-v2, which is designed to enhance the shortcomings which the sniffing program may evade detecting on a remote computer within a network segment, as is stated in Section 2. This protocol can also offer the detecting method on remote network across routers.

4.1 Overview

Basically, this protocol is designed to communicate between two entities where one entity is called TAPserver(or Responder) and the other is called TAP-client(or Initiator). We must also consider even that more than 2 computers are working on the network. Figure 1 shows that the TAP runs on the computers in LAN to detect whether any computer is in sniffing mode. It is assumed that the TAP-v2 has been installed on all the computers in the network. If a computer does not support the TAP-v2 and thus it cannot reply to the TAP-v2 messages when other computers requests it, it is recognized as compromised one. Assume that the computer named as TAP_1 wants to know whether any computer from TAP_2 to TAP_n is in sniffing mode. We assume also that a sniffing program is running in TAP_i, which is marked by a bomb symbol. Here TAP_1 sends a request message to all other computers one by one in order to detect the sniffing computer as the solid line arrows indicate. All other computers do the same operation as TAP_1. Here the check operation by TAP_n, for example, is shown by the dotted line arrows. The requested computers should send the reply message to the initiator of the TAP-v2.

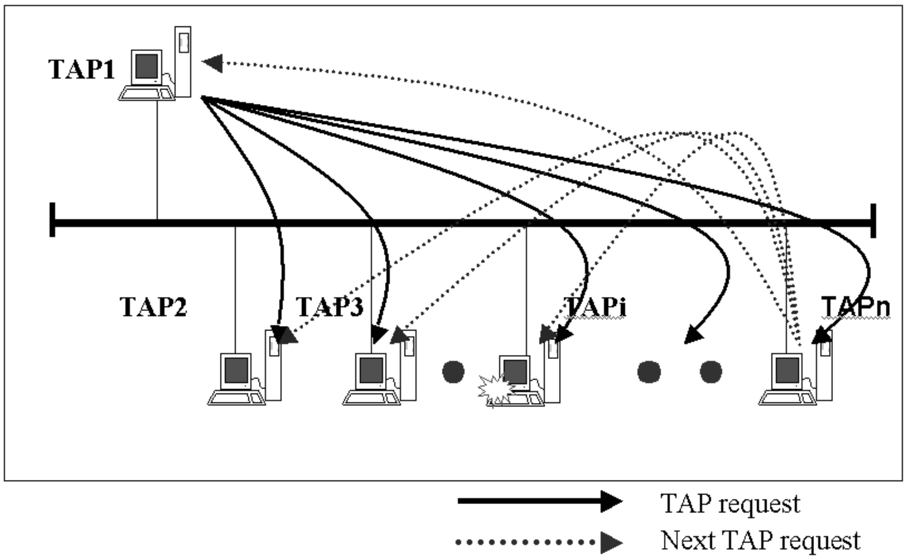

Fig. 1. Behaviors of TAP-v2

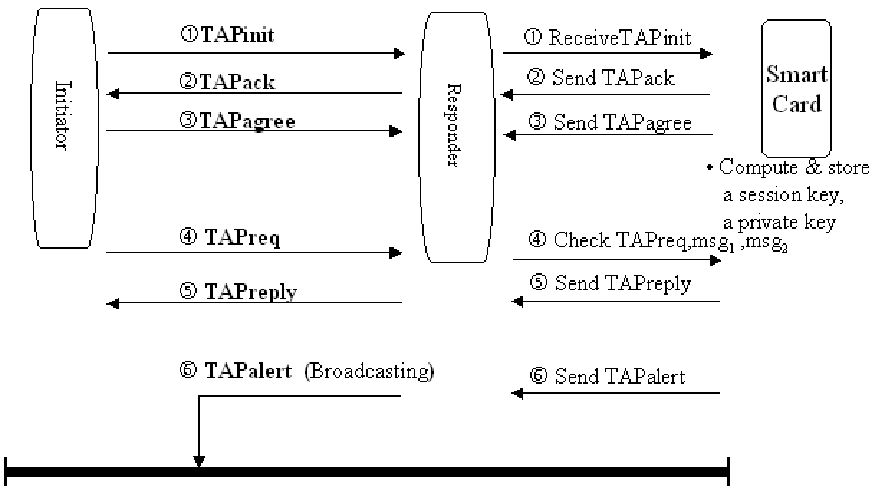

Fig. 2. The flow of TAP-v2 messages including smart card

4.2 Description without smart card

We here present the procedure employed for the TAP-v2. Figure 2 illustrates the flow of the messages with the TAP-v2.

STEP 1: Initialization

I $\longrightarrow$ R	:	**TAPinit**	(1)
R $\longrightarrow$ I	:	**TAPack**	(2)
I $\longrightarrow$ R	:	**TAPagree**	(3)

This step is for establishing a session key between two entities, ***I*** and ***R***, as a three-pass variation of the basic Diffie-Hellman key agreement protocol[5] called *STS using MAC*[11]. The detailed actions are as follows:

(1) ***I*** generates a secret random number x and sends ***R*** '**TAPinit**'.
(2) ***R*** generates a secret random number y and computes the shared key $K = g^{xy}$. ***R*** hashes the concatenated exponents ordered as in '**TAPack**', and sends '**TAPack**' to ***I***.
(3) ***I*** computes the shared key $K = g^{xy}$, and uses K to verify the received value as the hash value on the cleartext exponent received and the exponent sent in '**TAPinit**' message. Upon successful verification, ***I*** accepts that K is actually shared with ***R***, and sends '**TAPagree**' to ***R***. Finally, ***R*** verifies ***I***'s hash value. If ok, ***R*** accepts that K is actually shared with ***I***.

This step allows the establishment of a shared secret key between two entities with mutual authentication and explicit key authentication. Due to this, the communication of the two entities is protected from eavesdropping.

Without this step, in the case that NIC is in promiscuous mode we may have an opinion to give out the TAP-v2 alert message to the network. This idea constitutes *routine-interruptability* where ***R***(or TAPserver) is stopped not to eject '**TAPalert**' from the computer or anything else. Thus this step must be necessary.

Finally, it is assumed that the shared key is put in a private area of the entity such as smart card.

STEP 2: Request and reply

I $\longrightarrow$ R	:	TAPreq	(4)
R $\longrightarrow$ I	:	TAPreply	(5)

(4) If ***I*** wants to check the promiscuous NIC of ***R***, ***I*** sends '**TAPreq**' to ***R***. The '**TAPreq**' message is sent to each other one by one periodically. ***R*** should reply with '**TAPreply**'.

(5) If ***R*** receives '**TAPreq**' message, ***R*** should reply to the ***I*** with '**TAPreply**'. Since the checking routine has a shared secret key with ***I***, the routine can encrypt the checked result with the shared secret key after the routine hashes itself. Next, the routine sends '**TAPreply**' to ***R***. If ***R*** does not send '**TAPreply**' to ***I*** within a predetermined time limit or ***R*** sends an invalid '**TAPreply**' to ***I***, then ***I*** recognizes that ***R*** is running a sniffing program.

In case that ***R*** does not reply with '**TAPreply**', we should consider the following two cases:

Case1 The computer does not reply since it is dead.
Case2 The computer is compromised.

To distinguish two cases above, '*ping*' command is used. We can also distinguish two cases through examining the return value after sending '**TAPreq**'. When the '**TAPreply**' gets fraudulent, ***I*** broadcasts '**TAPalert**' to the network about the fact that a computer is compromised.

'**TAPreply**' message also constitutes to provide the detecting method on even the remote network. As this message is verified, it is possible to detect the tapping program on the remote network. From this reason, '**TAPreply**' is encrypted with the session key to provide confidentiality for the communication session.

STEP 3: Alert

$$\mathrm{R} \longrightarrow \text{Local Network} \quad : \quad \textbf{TAPalert} \qquad (6)$$

(6) If any message format is invalid or any computer identifies a promiscuous NIC, the computer advertises the '**TAPalert**' message on LAN by using the broadcast IP address or by using IP addresses of the requested computers.

In this step, a signature function is used for protecting data-forgery attack. If all the formats are valid and all the computers identify no promiscuous NIC, no '**TAPalert**' message is broadcast on the network. Suppose that a computer has already received '**TAPreply**' with 'Y', which means that a sniffing program is running. If the computer also receives '**TAPreply**' message with 'N' which means that a sniffing program is not running, the computer advertises the '**TAPalert**' message including 'N' on the network.

STEP 4: Verification

$$\boldsymbol{R} \longrightarrow \boldsymbol{O} \quad : \quad \textbf{TAPalert} \qquad (7)$$

(7) ***O*** means other computers, which receives '**TAPalert**' from ***R***. '**TAPalert**' includes ***R***'s signature.

This is the step for verifying that '**TAPalert**' is not forged. If '**TAPalert**' has been forged, this step will detect it and alert the users on the compromised ***R***. It is possible to check out if this message is forged as verifying ***R***'s signature. '**TAPalert**' is used only on LAN.

In addition, TAP-v2 possesses the following properties:

- If a computer wants to check the promiscuous NIC on the remote network, TAP-v2 can be processed by the same procedure. An attentive point is that ***I*** (or TAPclient) can not receive '**TAPalert**'. Thus '**TAPreply**' is used to identify the computer in promiscuous mode.
- In case that a user does not want to receive tapping alert message, it can turn off TAP-v2 by using the program supporting TAP-v2. However, the computer must still be capable of transmitting '**TAPreply**'.
- The initiation time of each computer can be changed manually or automatically in order not to congest the network with the TAP-v2 packets.
- TAP-v2 of each computer collects the host information with respect to host address automatically in a timely manner.

4.3 Description with smart card

In this section, we present the procedure employed for TAP-v2, with a smart card to make the shared session key secure. Figure 2 illustrates the flow of the messages using the smart card at responder-side. TAPclient (or I) does not employ the smart card because any computer on the remote network as well as LAN can become TAPclient.

STEP 1: Initialization

I $\longrightarrow$ R	:	**TAPinit**	
R $\longrightarrow$ SC	:	**TAPinit**	(1)
SC $\longrightarrow$ R	:	**TAPack**	(2)
SC $\longrightarrow$ R	:	**TAPagree**	(3)
R $\longrightarrow$ I	:	**TAPack**	
I $\longrightarrow$ R	:	**TAPagree**	

This step describes that a shared session key is built using a smart card. In the smart card, the computed session key is stored and keeps secure against the possible attacks discussed. Hereafter, **SC** denotes a smart card.

(1) ***R*** sends g^x of ***I*** to ***SC*** in order to build a shared session key between ***I*** and ***R***.
(2) ***SC*** computes the session key $K = g^{xy}$ using g^x and computes $hv_1 = h_K(g^y \parallel g^x)$. ***SC*** gives out g^y and hv to ***R*** to envelope them into '**TAPack**'. Of course, ***R*** will transmit '**TAPack**' to ***I***. Finally, the session key and private key are stored and made secure in ***SC***.
(3) ***SC*** sends $hv_2 = h_K(g^x \parallel g^y)$ to ***R***. This hv_2 is compared with '**TAPagree**' from ***I*** so that the session key can be confirmed.

STEP 2: Request and reply

$$\begin{array}{lcll} \mathrm{I} \longrightarrow \mathrm{R} & : & \mathrm{TAPreq} & \\ \mathrm{R} \longrightarrow \mathrm{SC} & : & \mathrm{TAPreq},\ msg_1 & (4) \\ \mathrm{SC} \longrightarrow \mathrm{R} & : & \mathrm{TAPreply} & (5) \\ \mathrm{R} \longrightarrow \mathrm{I} & : & \mathrm{TAPreply} & \end{array}$$

In this step, we describes how to use the session key stored in the smart card so that the encrypted message can be decrypted with the session key. To make up '**TAPreply**', msg_1 is also encrypted with the session key in the smart card.

(4) ***R*** sends $ENC_K(TimeStamp_I)$ to ***SC*** to be decrypted with the session key. Of course, the decrypted $TimeStamp_I$ is returned to ***R***. To make up '**TAPreply**', msg_1 is also sent to and encrypted in ***SC***. msg_2 is sent to and signed in ***SC***
(5) ***SC*** sends the encrypted msg_1 back to ***R***. After making up '**TAPreply**', this message is transmitted to ***I***.

STEP 3: Alert

$$\begin{array}{lcll} \mathrm{SC} \longrightarrow \mathrm{R} & : & \textbf{TAPalert} & (6) \\ \mathrm{R} \longrightarrow \text{Local Network} & : & \textbf{TAPalert} & \end{array}$$

This step describes '**TAPalert**' is signed with the private key stored in smart card.

(6) ***SC*** sends the signed $Sign_R(msg_1)$ back to ***R***. After making up '**TAPalert**', this message is broadcast to the LAN.

5 Implementation and Analysis

5.1 Overview

TAP-v2 has been implemented on Redhat LINUX version 7.0, kernel 2.2.17 in programming language C, using compiler gcc version 2.96.

To provide cryptographic primitives, a library *libicuc.a* programmed by the authors is used. To complete the implementation, some functions such as El-Gamal crypto primitives for signature function, Rijndael the winner of AES competition for block cipher, HMAC function and SHA for keyed hash function, etc., are prepared to this library.

Basically, the structure of TAP-v2 programs is divided into a client and a server. The client side is called TAPclient and the server side is called TAPserver. But only TAPclient works alone to provide the detecting method on the remote network. TAPserver does not work alone, but always works with TAPclient. That is to say, if TAPserver is on a computer, it means the computer has TAPclient together.

In this section, we present the TAP-v2 implemented on LINUX, not using smart card. The behavior of TAP-v2 is explained, and the server and client of TAP-v2 are also described in detail.

5.2 Module description

Behaviors. As mentioned before, the program model for TAP-v2 employs Client/Server model, which is the most commonly used paradigm in constructing distributed systems. For this reason, in order to provide the detecting method for all the computer on LAN, it is extremely natural that TAP-v2 programs should be installed on all computers.

Like the sniffing programs, TAP-v2 program, specially TAPserver, must also use the related functions to NIC for checking if the NIC is in promiscuous mode. Due to this, TAP-v2 program must be run by those ones to the root privilege.

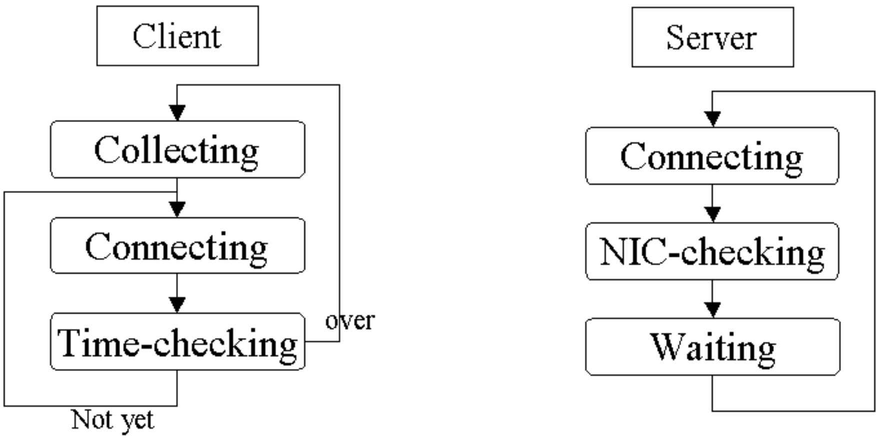

Fig. 3. Behaviors of TAP server/client

At the beginning of running TAP, to be divided into two roles as server and client, it calls fork() function that creates a new process (child process), which is an exact copy of the calling process (parent process). Next, to perform a role as tapping alert protocol, the following subsections explain how TAPclient and TAPserver behave on running, as shown in Figure 3.

TAPclient module

1. *Collecting step.* TAP-v2 start collecting the information of hosts on LAN such as host addresses, TAPserver working status on remote computer, etc. To collect the information from the network, tap_get_servinfo() is programmed using the return value of connect() call and ioctl() system call.
2. *Connecting step.* TAP-v2 tries to connect to other TAPservers through the list of host addresses gathered. If a TAPserver on a computer of the list is working, it checks if the computer is in sniffing mode through the secure communication session.

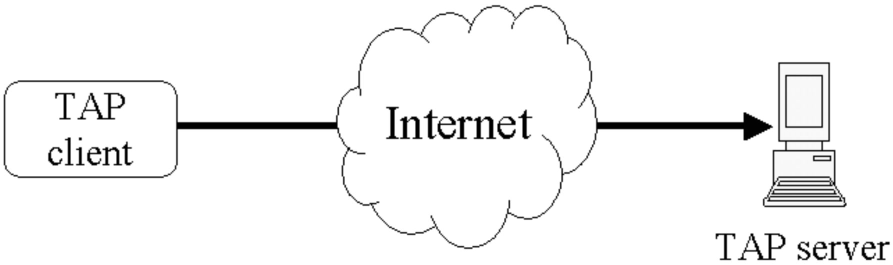

Fig. 4. TAPclient program working on a remote network

3. *Time-checking.* If it is the time to collect the host information, then the flow goes to the collecting step again. Otherwise, then runs from step matching step.

As shown in Figure 4, it is necessary to provide the detecting method on a remote network across routers or the Internet. Therefore, we have independently partitioned TAPclient from TAPserver and modified it into one program, performing a role as client on LINUX shell prompt. This program can be used to discover the status of NIC and needs host addresses as arguments.

TAPserver module

1. *Connecting step.* If TAPclient requests connecting for discovering the status of NIC, TAPserver should reply to the requests within the predetermined time, otherwise TAPclient consider this computer to be compromised.
2. *NIC-checking step.* TAPserver checks the status of NIC to discover if it is in promiscuous mode or not. This status is transmitted to TAPclient through the secure session. In this implementation, we have made the function check_nic() using ioctl() system call to see the status of NIC.
3. *Waiting step.* After the end of checking, TAPserver enters waiting step to receive other requests for connecting.

Unlike TAPclient, TAPserver always works with TAPclient when TAP-v2 operates on a computer system. In addition, TAPserver module must be run by the root since it should check the status of NIC.

5.3 Analysis

Security analysis. After TAP-v2 is initialized, suppose that an attacker wants to stop the routine. In this case a computer will send '**TAPalert**' out on the network. Other computers in the network will then recognize that the computer is malicious, and thus alert the users.

An attacker may want to forge the routine so that the forged routine can conduct as the original one. Note, however, that even though the attacker may be aware of the hash value of $\boldsymbol{R}$, the attacker cannot generate the signature value if it does not know the shared secret key of $\boldsymbol{I}$ and $\boldsymbol{R}$. Due to this, any attacker cannot forge the routine.

If any attacker also wants to forge the data from the checking routine, the attacker should get the shared secret key of $\boldsymbol{I}$ and $\boldsymbol{R}$ because the data is encrypted with a shared secret key. Since the shared secret key of $\boldsymbol{I}$ and $\boldsymbol{R}$ is put in a private area like smart card, it is very difficult for the attacker to get the key. Therefore, the attacker can hardly forge the data being sent.

Comparisons. IDS (Intrusion Detection System) [18, 17] is a famous system which detects any intrusion on the network. In comparison with the TAP-v2, IDS does not check the NIC on the remote computer but analyzes the packets on the network. Therefore, IDS cannot check the promiscuous mode of the NIC on a remote computer.

Unlike the proposed TAP-v2, *IFStatus* [16] and *CPM* [15] provide the method to discover if NIC is in promiscuous mode inside local computer only. For checking NIC, these tools show the result similar to a built-in Unix command '*ifconfig*'.

To our knowledge, the tools to have the ability that checks the status of NIC are known as *Antisniff* and *Neped*. Unfortunately, they does not provide the checking method to discover the status of NIC on the remote network across routers.

In comparison with these tools, Table 1 displays the TAP-v2 is much better except the fact that TAP-v2 must be installed on all the computers. The TAP-v2 allows a system to check if a sniffing program is run on a remote network as well as on a remote computer within a network segment. It was previously impossible to check if a sniffing program is run in a remote network. Unlike other tools - *Antisniff* and *Neped*, TAP-v2 is not affected by OS kernel patch and does not create a large amount of network traffic similar to DoS(Denial of Service) attack. If TAP-v2 is also used, there is no method to be evaded by any sniffing program since the security features of cryptographic primitives make the TAP-v2 secure. In the meanwhile, if other tools like *Antisniff* and *Neped* are used, there exist a few methods to be evaded. Therefore, we think that TAP-v2 is much more powerful mechanism compared to others in detecting the tapping.

Table 1. Comparisons with other tools - *Antisniff*, *Neped*

Classifications	TAP-v2	Others
Applicability on remote network	Yes	No
Sensitivity to OS kernel patch	No	Yes
Possibility of DoS attack	No	Maybe
Possibility of evasion	No	Yes
How many computers to install?	All	One

Traffic Overhead. Since additional packets are generated periodically for supporting TAP-v2 on the network, this may slightly degrade the network performance if the network bandwidth has already been fully utilized for regular traffic. This is the only overhead incurred in the implementation of TAP-v2. For this reason, here we measure the traffic overhead of TAP-v2.

The number of packets is proportionally generated to n^2,where n is number of hosts on a network. When a host is waiting until the next TAP is performed on the host, it is called *waiting time*, following a uniform distribution. Thus,on the network, the number of packets is given by the following equation [19]:

$$p = \kappa \frac{n^2}{t} \tag{1}$$

,where p is the number of packets, t is waiting time and κ is a proportional constant. If t is given in second and κ in bytes/second, then p is in unit of bytes/sec. To get a constant κ, we need to refer to Table 2 and the processing time from '**TAPinit**' to '**TAPalert**'. Practically, the processing time is known as 10 seconds on a computer with 800MHz CPU. If the header length of TCP/IP[7, 8] is included, κ becomes $92.2 = (646 + 276)/10$. Therefore, we can obtain the the following equation:

$$p = 92.2 \frac{n^2}{t} \tag{2}$$

TAP-v2 has been implemented based on TCP[8] as a network communication protocol. If UDP[9] instead of TCP is employed, κ gets $85.0 = (646 + 204)/10$. The traffic overhead caused by TAP-v2 can be slightly reduced on the network by using UDP.

Table 2. The length of TAP messages

TAP messages	Length(Bytes)
TAPinit	74
TAPack	158
TAPagree	30
TAPreq	26
TAPreply	42
TAPalert	40
Total	646

Figure 5 shows the traffic overhead depending on the number of hosts on the network. It appears naturally that the more the waiting time, the less the number of packets.

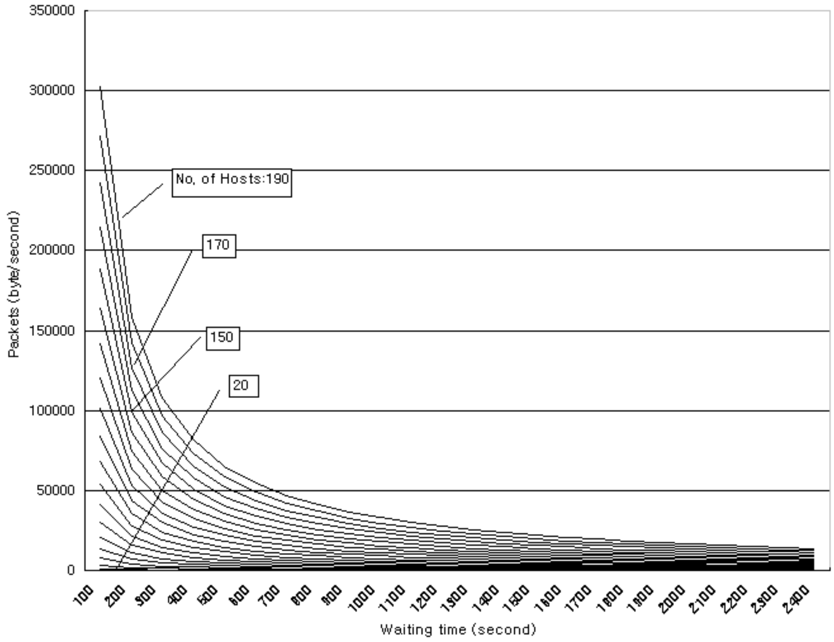

Fig. 5. Traffic overhead for TAP

6 Conclusions

In this paper, we have presented the design and implementation of a new protocol that provides the network with the capability of checking if the NIC in a local or remote computer. This protocol is also designed to provide the method that can check the status of NIC even the computer on remote network.

The protocol is based on the cryptographic primitives with authentication, integrity and confidentiality. To hide the session key and private key, smart card is also employed, which has the built-in CPU and memory independently. Due to these, it is very difficult for an attacker to forge the checking routine and the data being sent. In addition, although the checking routine is stopped, the other computers on the network can recognize the computer has been compromised.

In comparison with other tools like *Antisniff* and *Neped*, TAP-v2 has the good properties where this protocol is not affected by OS kernel patch, is not called DoS attack, and it is no way to evade this protocol.

If you, however, want the computer to be protected from sniffing attack on the network, TAP-v2 must be installed on all the computer which you want to protect. Otherwise, the rest computers that TAP-v2 is not installed on should be recognized as being compromised.

In fact, we have not implemented TAP-v2 using a smart card, just describes the design for the smart card. Therefore, we need to add the smart card to our implementation. In addition, we think that our current implementation should be much updated to raise the running performance.

Finally, if TAP-v2 should be installed on most of computers in the Internet, sniffing or tapping attack in the network based on TCP/IP will be disappeared.

References

1. G.W. Ahn, K.K. Kim, and H. Y. Youn, "Tapping Alert Protocol", *Proc. of WET-ICE2000 Workshop on Enabling Technologies*, NIST, USA, IEEE Computer Society, June, 2000.
2. "Antisniff", http://www.l0pht.com/antisniff/
3. "AntiSniff finds sniffers on your local network", http://www.securiteam.com/tools/.
4. "CERT Advisory CA-1994-01 Ongoing Network Monitoring Attacks", http://www.cert.org/advisories/CA-1994-01.html, Feb., 1994.
5. W. Diffie, M. Hellman, "Directions in Cryptography", *IEEE Transactions on Information Theory*, 22, pp. 44-654, 1976.
6. FIPS 180-1, "Secure Hash Standard", *NIST*, US Department of Commerce, Washington D.C., April 1995.
7. J. Postel, "Internet Protocol", RFC 760, *USC/Information Sciences Institute*, Jan. 1980.
8. J. Postel, "Transmission Control Protocol", RFC 761, *USC/Information Sciences Institute*, Jan. 1980.
9. J. Postel, "User Datagram Protocol", RFC 768, *USC/Information Sciences Institute*, Aug. 1980.
10. R. Ankney, D. Hohnson and M. Matyas, "The Unified Model", contribution to X9F1, October 1995.
11. S. Blake-Wilson and A. Menezes, "Unknown Key-Share Attacks on the Station-To-Station (STS) Protocol", *Technical report CORR 98-42*, University of Waterloo, 1998.
12. C.P. Schnorr, "Efficient signature generation by smart cards", *Journal of Cryptology*, pp161-174, April 1991.
13. J. Daemen and V. Rijmen, "AES proposal: RIJNDAEL", NIST publications, March 1999.
14. U. Hansmann, M.S. Niclous, and *ex. al.*, "Smart Card Application Development Using Java", *Springer Verlag*, 2000.
15. "Check Promiscuous Mode", ftp://ftp.uu.net/pub/security/cpm/cpm.1.2.tar.gz
16. ftp://coast.cs.purdue.edu/pub/unix/tools/ifstatus/ifstatus.tar.gz
17. B. Mukherjee, L. T. Heberlein and K. N. Levitt, "Network Intrusion Detection", *IEEE Network Volume*: 83, pp. 6-41, May-June 1994.
18. G.G. Helmer, J.S.K. Wong, V. Honavar and L. Miller, "Intelligent Agents for Intrusion Detection", *IEEE Information Technology Conference*, pp.121-124, 1998.
19. A. Leon-Garcia, "Probability and Random Processes for Electrical Engineering", *Addison Wesley*, pp. 101.