# Tapping Alert Protocol

Gookwhan Ahn, Kwangjo Kim, and Hee Yong Youn
*Information and Communications University*
*Taejon, Korea*
{*tachyon, kkj, youn*}*@icu.ac.kr*

## Abstract

*This paper proposes a new protocol called tapping alert protocol* (TAP)*, which detects if any computer is eavesdropping on the network. When tapping is detected, it alerts the network users on the danger of tapping. In designing the protocol, a model for an attacker is developed who sniffs the network and breaks down the proposed protocol, and the requirements on the protocol in terms of efficiency and security 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 schemes to guarantee authentication of the code monitoring tapping operation and integrity of the data being sent. Finally, the proposed protocol can protect authorized users from illegal eavesdropping on the network.*

Keywords: cryptography, network security, promiscuous mode, sniffing, tapping alert protocol.

## 1. Introduction

Numerous techniques have been devised that allow attackers to 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 a new protocol, which protects the system from tapping. It provides a local area network with a mechanism by which it can detect if any computer is eavesdropping on the network. Upon detection, it alerts the network users on the danger of tapping that can leak the information of the users.

Tapping has been called by several different terminologies - snooping, sniffing, eavesdropping, etc. Tapping in a local area network is relatively easy since Ethernet is a shared network and all the packets on the network are broadcast. 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 the corresponding passwords are extracted through tapping, the person eavesdropping 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 is possible 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 [1]. Of course, when a network administrator needs to analyze the network performance, the promiscuous mode is also used. However, it can still collect the host and user authentication information in addition to the data on network operation, which can be used for tapping.

There are some countermeasures developed to avoid tapping as follows [1].

- Do not use the NICs that support 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 [9,10].

Previously, to the best of our knowledge, it has been possible to detect the promiscuous mode of only the computers on LAN with the countermeasures above. If someone is running a sniffing program on a remote computer system, nobody could detect it. For this reason, we propose a new method for detecting the promiscuous mode even on remote computers. The new method, called TAP (Tapping Alert Protocol), uses cryptographic schemes [2,3] in order to provide authentication of module checking and integrity of data.

The rest of the paper is organized as follows. Section 2 discusses the background issues related to tapping. In Section 3, the proposed protocol is presented which prevents the local area network from being tapped by any attacker. We analyze the protocol in Section 4. Finally, conclusions are given in Section 5.

## 2. Background

In this section, we first present the requirements on the protocol detecting promiscuous mode. Next, we describe how to check if an NIC is in sniffing mode. We also examine the types of attackers breaking into the network with some sniffing tools and build assumptions by which the attackers break down the proposed protocol.

### 2.1. Requirements on the Protocol

Here we do not consider how to prevent the root permission from being compromised. There are many techniques protecting a computer from the attack on the root permission through Internet. Under this assumption, we consider the requirements on the protocol in terms of security and efficiency.

**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 also necessary.

The main objective of public-key encryption is to provide privacy or confidentiality. However, public-key encryption alone does not provide either data authentication or data integrity. Therefore, in building a scheme handling tapping, some additional mechanisms need to be employed in addition to public-key encryption scheme in order to provide authentication and integrity. They are message authentication code (keyed hash function) and signature function [2,7].

We also need to employ the three-pass variation of the basic Diffie-Hellman protocol to provide a shared secret key between two entities involved when a message authentication code is used [13].

**Efficiency requirements**

- *Low congestion*: If any computer runs a tapping program, this information needs to be broadcast so that other computers in the network can be aware of the fact. In the case that there exist many computers in the network, this may cause the network congested. In order to avoid this problem, the scheme developed for handling tapping needs to use UDP or IP address for broadcast in the network [4-6].

### 2.2. Detecting the Promiscuous Mode

Here we present how to detect if an NIC operating under Unix is in promiscuous mode. The following C code describes the routine for NIC running in SunOS 4.x [9]. If the value of the underlined *if*-statement 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 network interface is in promiscuous mode */

    }
}
```

### 2.3. Possible Attacks

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

- The attacker obtains the root privilege in order to eavesdrop the network or stop the protocol. If an attacker has no root permission on the computers, it cannot tap the network.

- The attacker forges the routine checking the network interface. 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 routine does, and the network users and computers will misjudge that it is a legitimate one.

- The attacker forges the data sent to other computers. Here if an attacker wants to provide the network with the routine checking the network interface and wants it to run normally but still wants to deceive the network users and computers, it forges the data sent.

We next present the proposed protocol.

## 3. The Proposed Protocol

In this section we present a new protocol preventing a network from tapping. We describe the notations, message formats, and message flow supported by the proposed tapping alert protocol (TAP).

The TAP is designed to alert other computers to the danger of sniffing in the local area network if any computer runs a sniffing program in the environment of a shared network. Until now, there has been no method checking if a computer in a remote network is running a sniffing program. With the proposed TAP, the promiscuous NICs running sniffing program remotely can be detected.

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

### 3.2. Message Formats

We design a set of message formats as follows:

- $msg_n$ : messages transmitted among the entities

$msg_1=(h_K(Routine()) \| \text{'Y' or 'N'} \| IPaddr_R \| TimeStamp_I \| TimeStamp_R \| MachineTypes_R)$

$msg_2= \text{'Y' or 'N'} \| IPaddr_R \| TimeStamp_R \| MachineTypes_I$

- **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***.

  $(\text{'TAPinit'} \| g^x)$

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

  $(\text{'TAPack'} \| g^y \| g^x \| h_k(g^y \| g^x)\,)$

- **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 the two entities share a session key without any attack.

  $(\text{'TAPagree'} \| h_k(g^x \| g^y)\,)$

- **TAPreq**: Query message for requesting a check of a computer, which may be running a sniffing program.

  $(\text{'TAPreq'} \| TimeStamp_I\,)$

- **TAPreply**: Reply message to the checking routine above.

  $(\text{'TAPreply'} \| ENC_K(msg_1)\,)$

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

  $(\text{'TAPalert'} \| Sign_I(msg_2) \| msg_2\,)$

- **TAPreqpk**: To verify '**TAPalert**', other computers on the network request the public key of entity ***I*** that has broadcast '**TAPalert**'

  $(\text{'TAPreqpk'} \| TimeStamp_O\,)$

- **TAPackpk**: This is sent to the computer that has requested '**TAPreqpk**'.

  $(\text{'TAPackpk'} \| PK_I \| TimeStamp_O \| TimeStamp_I\,)$

## 3.3. Message Flow

### 3.3.1 Overview

Figure 1 shows that the TAP runs on the computers on a local area network to detect whether any computer is in sniffing mode. It is assumed that the TAP has been installed on all the computers on the network. If a computer does not support the TAP and thus it cannot reply to the TAP messages when other computers requests it, it is recognized as a one being compromised.

In the figure, assume that the computer marked as `TAP1' wants to know whether any computer from TAP2 to TAPn is in sniffing mode. We assume also that a sniffing program is running in TAPi, which is depicted by a bomb symbol. Here TAP1 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 TAP1. Here the check operation by TAPn, for example, is shown by the dotted line arrows. The requested computers should send the reply message to the initiator of the TAP.

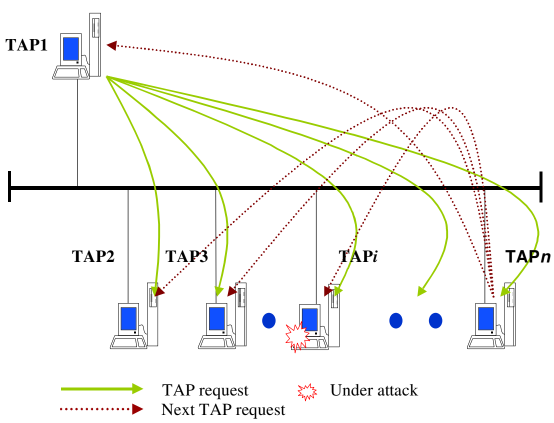


Figure 1. The TAP operation among computers.

### 3.3.2 Procedure

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

**STEP 1: Initialization**

$$I \rightarrow R : \textbf{TAPinit} \quad (1)$$
$$R \rightarrow I : \textbf{TAPack} \quad (2)$$
$$I \rightarrow R : \textbf{TAPagree} \quad (3)$$

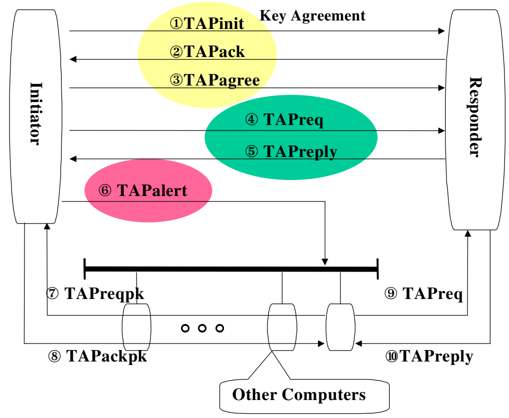


Figure 2. The flow of TAP messages.

This step is for establishing a session key between two entities, ***I*** and ***R***, as a three-pass variation of the basic Diffie-Hellman protocol. When a computer boots up, it is initiated for just once. 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 $\boldsymbol{K = g^{xy}}$. ***R*** hashes the concatenated exponents ordered as in '**TAPack**', and sends '**TAPack**' to ***I***.

(3) ***I*** computes the shared key $\boldsymbol{K = g^{xy}}$, 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 successful, ***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.

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**

$$I \rightarrow R : \textbf{TAPreq} \quad (4)$$
$$R \rightarrow I : \textbf{TAPreply} \quad (5)$$

(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 initiator 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:

- One is the case that the computer does not reply since it is dead.
- The other case is that the computer is compromised.

To distinguish the two cases above, 'Ping' command is used. To be sure, we assume that the TAP is implemented in the OS kernel such as TCP/IP stack.

**STEP 3: Alert**

$I \rightarrow$ Local Network **: TAPalert** (6)

(6) If any message format is invalid or any computer identifies a promiscuous NIC, the computer advertises the '**TAPalert**' message on the local area network by using the broadcast IP address.

In this step, a signature function is used for precluding 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**

$O \rightarrow I$ : **TAPreqpk** (7)
$I \rightarrow O$ : **TAPackpk** (8)
$O \rightarrow R$ : **TAPreq** (9)
$R \rightarrow O$ : **TAPreply** (10)

(7) ***O*** means that other computers received '**TAPalert**', and ***O*** requests ***I***'s public key to verify '**TAPalert**' including the signature value.
(8) ***I*** sends its public key to ***O***.
(9) ***O*** also sends ***R*** 'TAPreq' to verify that the '**TAPalert**' value is 'Y'.
(10) ***R*** replies to ***I*** with '**TAPreqply**', but do not advertise the '**TAPalert**' message.

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 ***I.***

In addition, the TAP possesses the following properties:

- If a computer wants to check the promiscuous NIC on the remote network, the TAP can be processed by the same procedure.
- In case that a user does not want to receive tapping alert message, it can turn off the TAP by using the program supporting the TAP. 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 packets.

## 4. Analysis of TAP

In this section, we first analyze the performance of the proposed TAP in terms of security. The merits and demerits of the TAP are then discussed. The implementation issue is also discussed.

### 4.1 Security Analysis

After TAP is initiated, 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 ***R***, the attacker cannot generate the signature value if it does not know the shared secret key of ***I*** and ***R***. Due to this, any attacker cannot forge the routine.

If any attacker wants to forge the data from the checking routine, the attacker should get the shared secret key of ***I*** and ***R*** because the data is encrypted with a shared secret key. Since the shared secret key of ***I*** and ***R*** is put in a private area, however, it is very difficult for the attacker to get the key. Therefore, the attacker can hardly forge the data being sent.

### 4.2. Merits and Demerits

The proposed TAP allows a system to check if a sniffing program is run on a remote computer. It was previously impossible to check if a sniffing program is run in a remote computer.

IDS (Intrusion Detection System) [11,12] is a famous system which detects any intrusion on the network. In comparison with the TAP, IDS does not check the network interface on the remote computer but analyzes the packets on the network. Therefore, IDS cannot check the promiscuous mode of the network interface on a remote computer.

Since additional packets are generated periodically for supporting the TAP, 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 the TAP.

### 4.3. Implementation Issues

Figure 3 shows a structure where the TAP is assumed to be implemented under Linux platform. Here a routine in the Linux kernel need to keep in close contact with the routines of other computers for authentication and integrity using public-key encryption and keyed hash function. When the TAP identifies tapping, it broadcasts an alert message to the network by using broadcast IP address.

To insert the TAP into the kernel and check promiscuous mode, we update the *dev_set_promiscuity*() function of linux kernel. Since the TAP is implemented in the kernel like TCP/IP stack, there is no way to kill the TAP without killing the kernel.

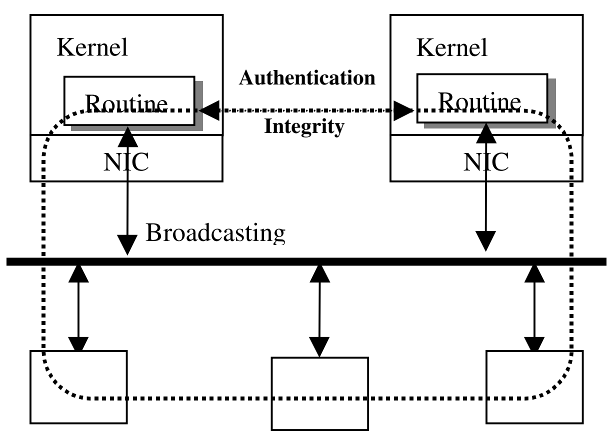


Figure 3. Implementation with Linux kernel.

## 5. Conclusion

In this paper we have presented a new protocol that provides the network with the capability of checking if the network interface in a local or remote computer is tapping. The protocol works based on a hash function and a public key scheme with the properties of authentication and integrity. Due to this, it is very difficult for an attacker to forge the checking routine and the data being sent.

We will implement the TAP in Linux as a future research.

## Acknowledgements

The authors would like to thank Prof. Ben Lee at ICU for his proofreading of this paper.

## References

[1] http://www.cert.org/ftp/cert_advisories/CA-94:01.network.monitoring.attacks, Feb. 1994.

[2] W. Diffie, M. Hellman, “New Directions in Cryptography," IEEE Transactions on Information Theory, 22, pp. 44-654, 1976.

[3] FIPS 180-1, Secure Hash Standard, NIST, US Department of Commerce, Washington D.C., April 1995

[4] Postel, J.,"Internet Protocol," RFC 760, USC/Information Sciences Institute, Jan. 1980.

[5] Postel, J., "Transmission Control Protocol," RFC 761, USC/Information Sciences Institute, Jan. 1980.

[6] Postel, J., "User Datagram Protocol," RFC 768, USC/Information Sciences Institute, Aug. 1980.

[7] Alfred, J. M., Paul, C. O., Scott, A. V, “Handbook of Applied Cryptography,” CRC press, pp.25-28, 1997.

[8] Douglas, R. S., “Cryptography Theory and Practice,” CRC Press, pp. 232-236, 1995.

[9] http://www.certcc.or.kr/tools/network/cpm/cpm.1.2.tar.gz

[10] ftp://coast.cs.purdue.edu/pub/unix/tools/ifstatus/ifstatus.tar.gz

[11] Mukherjee, B., Heberlein, L. T., Levitt, K. N. “Network Intrusion Detection,” IEEE Network Volume: 83, pp. 6–41, May-June 1994.

[12] Helmer, G.G., Wong, J.S.K., Honavar, V., Miller, L. “Intelligent Agents for Intrusion Detection,” IEEE Information Technology Conference, pp.121-124, 1998.

[13] W. Diffie, P. van Oorschot and M. Wiener, “Authentication and Authenticated Key Exchange,” Designs, Codes and Cryptography, 2, pp.107-125, 1992.