高品質インターネット　5－14
(2002. 10. 11)

Security on the Secure Shell

齋藤　孝道†　　鬼頭　利之††

† 東京工科大学 工学部 情報工学科　〒192-0982 東京都八王子市片倉町 1404-1
†† 東京理科大学大学院　〒278-8501 千葉県野田市山崎 2641
E-mail: †saito@cc.teu.ac.jp

Security on the Secure Shell

Takamichi SAITO† and Toshiyuki KITO††

† Tokyo University of Technology, Katakura 1404-1, Hachioji, Tokyo, 192-0982, Japan.
†† Science University of Tokyo, Yamazaki 2641, Noda, Chiba, 278-8501, Japan.
E-mail: †saito@cc.teu.ac.jp

Abstract Although some flaws have been found out in the SSH, the Secure Shell, there is not so much discussion about its architecture or design safety. Therefore, in this paper, considering the SSH's architecture, e.g. the key exchange protocol, the user authentication protocols and its total design of the SSH, we not only discuss the SSH's architectural safety but show some critical flaws for SSH users. For establishing the SSH connection, before the user authentication, the SSH sever and client are exchanging a session key, which can make secure communication. Then, over the secret channel encrypted by the session key, the SSH server are authenticating a user in the SSH client using with a user's password or public key. However, owing to the defects in the SSH protocols and its design, a user can be deprived of its password in the authentication protocol. Moreover, we will show that those who uses its public key for authentication are exposed to the menace same as password-oriented users are.

Key words System Security, Authentication, Authentication Protocol

1. Introduction

SSH, the Secure Shell [1] ～ [6], is widely used as a secure remote terminal software. The SSH can make us login to a remote computer over insecure networks, execute a command, and transfer a file between a remote computer and a local computer. As a composition of the system, the SSH software is composed of the *SSH server* and the *SSH client*. There are two version, i.e. SSH1 and SSH2, which are specified in the some drafts [2] ～ [6]. Going through the *protocol version exchange* phase, the *key exchange* phase, and the *user authentication* phase in this order, the SSH sever and client in both SSH1 and SSH2 establish a secure communication channel. This SSH's secure channel is considered to guarantee that the a user can communicate with the intended SSH server secretly.

After the protocol version exchange phase, in the key exchange phase, the client confirms the server, and the SSH sever and client are exchanging a session key, which can make them utilize an encrypted communication. And then, over the encrypted communication channel with the session key, the SSH server authenticates the user in the SSH client. Note that there executes to exchange the session key before the user authentication. In a word, the SSH server can't identify the SSH client in the user authentication phase. Therefore, depending on case, there is possibility that the SSH server can be deceived by the malicious client.

In this paper, we will consider the structure of the SSH, provide the defects of the SSH in the key exchange phase, the user authentication phase, and these architectural combination. And more, for confirming in the practical point of view, we referred to SSH version 1.2.32, SSH version 3.0.2, and OpenSSH version 3.0.2 as the implementation.

2. Background

2.1 SSH Keys

There are four types of keys used in SSH: *Host Key*, *Server Key*, *User Key*, and *Session Key*.

Host Key : All SSH server should have a host key, i.e. a public key. The host key is used in key exchange protocol to verify that the client is really talking to the intended server. For this purpose, the client must have a priori knowledge of the server's host key.

Sever Key : A temporary, public key used only in the SSH1. It is created by the server at regular intervals (by default every hour) [1] and protects the *session key*.

User Key : A user key is used for the public key authentication by the SSH server. All implementations of SSH must support the public key authentication.

Session Key : A session key is a symmetric key generated in each session. It is used for encrypting the communication between the SSH client and server, and should be shared properly by the key exchange method.

2.2 Notations and Terms

Principals : The SSH is composed of the server and client. The *SSH server*, represented by S, is a principal executing the SSH server program, and the *SSH client*, represented by C, is a principal executing the SSH client program. Moreover, the intruder or the malicious user is represented by I.

Messages : Let Msg be an arbitrary message. In the case that the client C transmits a message Msg to the server S, we write the following: $C \rightarrow S : Msg$

And, the case that the client C and the server S are mutually exchanging a message Msg is described as follows: $C \leftrightarrow S : Msg$

Moreover, we introduce expressions about the intruder pretending the other principal. For example, in case that the intruder I masquerading the server S transmits a message Msg to the client C, we write the following: $I(S) \rightarrow C : Msg$

Encryption and Keys : We write the expression $\{Msg\}_Y$ to denote the resulting of encrypting a message Msg with a key Y, and $\{Msg\}_{Y^{-1}}$ that of signing Msg with a private key Y^{-1} which is correspoding to a public key Y. In the case that a message Msg is encrypted with keys Y and Z in this order, we write $\{\{Msg\}_Y\}_Z$

On the other hand, the key P_S denotes the SSH server S' *host key*, the key H_S the SSH server S' *server key*, and the key P_C the public key of a user on the client C. KS_{CS} or KS_{SC} is the session key between S and C.

Key Exchange in SSH2 : p is a large safe prime, g is a generator for a subgroup of $GF(p)$, q is the order of the subgroup. V_S is S' version string, and V_C is C's version string. I_C is C's key exchange message, and I_S is S' key exchange message. Random number y, x $(1 < x, y < q)$ are generated S and C respectively. K_S is $g^y \bmod p$, K_C is $g^x \bmod p$. Moreover, K_{CS} is $g^{xy} \bmod p$ as a seed of session key. In this paper, this K_{CS} itself is considered as a session key.

User Authentication : *modulus* denotes a part of arbitrary public key, it is for the identifer of the public key. *challenge* represents a 256-bit random number created by S. *username* and *password* denotes a user's login-name and its password respectively. Especially, when we write $modulus_C$, $username_C$, $password_C$, they are denoted to be used in C respectively.

Miscellany : SA denotes lists of encrypting algorithms, compressing algorithms, authentication ways, and so on. And, RN represents a random number as a cookie. S_{CS} or S_{SC} is the session identifer between S and C. The expression $f(Msg_1)$ represents a hash value obtained from Msg_1. Similarly, $f(Msg_1, Msg_2)$ shows a hush value obtained from the concatenation of Msg_1 and Msg_2. Ack denotes an arbitrary messages for acknowledgement.

3. SSH, Secure Shell

3.1 Architecture

Between the SSH server and client, these following three phase in this order are executed to establish the connection: *protocol version exchange*, *key exchange*, and *user authentication*.

Firstly, in the *protocol version exchange* phase, the SSH server and client notify their SSH version, such as `"SSH-1.5-1.2.22"`, and confirm whether they support the version .

Nextly, in the *key exchange* phase, the server sends the host key, i.e. its public key, which can make the client identify the server, and also sends session parameters. Especially in SSH1, the server also sends the server key with them. On the assumption that the client obtains proper host keys before its connection, it can confirm the server with the host key. When it can't match the host key in its database, a user must decide to select if it is accepted or not. If a user rejects it, the connection ends. Moreover, the most important thing is that the session key and session identifer are exchanged in this phase. The SSH1 client creates the session key, encrypt it with the server key and the host key, and send it to the SSH1 server. On the other hand, in SSH2, the session key is shared by the modified Diffie-Hellman key exchange, which is represented as *diffie-hellman-group1-sha1*

in [4]. Another way, not supported in the most version, is specified in [7]. Moreover, the session identifer is made by the exchanges (explaining later).

Finally, in the *user authentication* phase, the SSH server authenticates a user in the SSH client. There are some supported authentication methods, differed in SSH1 and SSH2. Here, the SSH server and client must agree an authentication method. Note that, since the connection ends if both of them can't agree, the SSH server can compel the client to agree an arbitrary method as the server wants. Which means that, even those who wants to utilize the public key authentication can be forced to use the password authentication if the user wants to continue the session.

The followings are user authentications mainly supported in SSH1 and SSH2 :

Password Authentication : Password authentication is a method to confirm if the user can show the corresponding secret, i.e. password. So, utilizing this authentication, the user should type its password to the SSH client.

Public Key Authentication : Public key authentication is a method to confirm if the user holds the corresponding private key. There exists two kinds of this confirmation: the *challenge and response scheme*, and the *digital signature*.

Rhosts Authentication : For r-commands, `/etc/hosts.equiv` and `.rhosts` are used to decide if trusted-host or not. In this paper, since this method are not considered secure, it is omitted to deal with.

Rhosts with Public Key Authentication : For same purpose of the below, the SSH provides this method added with public key authentication of client. In this paper, it is also omitted to deal with.

3.2 Key Exchange Phase

In this section, the key exchanging procedures are explained in details. Although there are differences in SSH1 and SSH2, both servers can be identified by the client in this phase.

3.2.1 Key Exchnage in SSH1

M1) $S \rightarrow C : P_S, H_S, SA, RN$
M2) $C \rightarrow S : SA, RN, \{\{KS_{CS}\}_{P_S}\}_{H_S}$
M3) $S \rightarrow C : \{ack\}_{KS_{CS}}$

These above protocol in SSH1 can make the server and client share the session key and its session identifer, and can make the client authenticate the server:

After the client connection request and protocol version exchange, firstly, the SSH server S sends, to the client C, the host key P_S, the server key H_S the session parameter SA, and the random number RN (**M1**). The client C confirms whether the received host key P_S is same as one in its database. If both aren't identified as the same, the client program warns to the user that the host key is changed, and the user should decide if accepting it or not. And then, both sides compute the session identifer S_{CS}, which is the MD5 hash value of a concatenation of RN, P_S, and H_S. Moreover, the client C randomly creates the session key KS_{CS} that the server and client support. After encrypting the session key with the host key and the server key, the client sends it to the server, with SA and RN (**M2**). This encrypted message can be decrypted only by the proper server. Finally, by receiving an acknowledgement encrypted with the KS_{CS}, the client can decide that the session key is shared with the server properly (**M3**).

3.2.2 Key Exchange in SSH2

M1) $S \leftrightarrow C : SA$
M2) $C \rightarrow S : K_C$
M3) $S \rightarrow C : K_S, P_S,$
$\{f(V_C, V_S, I_C, I_S, P_S, K_C, K_S, KS_{CS})\}_{P_S^{-1}}$
M4) $C \rightarrow S : \{KS_{CS}\}_{KS_{CS}}$

These above protocol in SSH2 can make the server and client share the session key and session identifer. This protocol is called *deffie-hellman-group1-sha1*, and can make the client authenticate the server: Firstly, the SSH server S and client C negotiate some session parameters with SA (**M1**). Next, the client C creates a random number x, and computes K_C $(= g^x \bmod p)$, sends it to the server S (**M2**). After receiving it, the server S generates a random number y, computes the session key KS_{CS} $(= g^{xy} \bmod p)$. And, it also computes H, which is the session identifer S_{CS}, a hash value of a concatenation of V_C, V_S, I_C, I_S, P_S, K_C, K_S, and KS_{CS}. Moreover, the server tranmits $K_S (= g^y \bmod p)$, its host key P_S, and the hash value H signed with its private host key P_S^{-1}. When the client receives them, using by its database, it confirm if P_S^{-1} really is the host key correspoding to the server S. And then, the client also computes the session key KS_{CS} $(= g^{xy} \bmod p)$, the hash value H, and verifies the signature of H with its host key P_S (**M3**). Finally, the server and client confirm the session key is shared properly (**M4**).

3.3 User Authentication Phase

So far, since the SSH server and client share the session key KS_{CS} and the session identifer S_{CS}, they can communicate over the *secure* channel encrypted with the key. However, although the server is identified by the client, the server doesn't have a confirmation of the client or user. Therefore, there needs to authenticate the user.

3.3.1 Password Authentication in SSH1

M1) $C \rightarrow S : \{username\}_{KS_{CS}}$
M2) $S \rightarrow C : \{ack_1\}_{KS_{CS}}$
M3) $C \rightarrow S : \{password\}_{KS_{CS}}$
M4) $S \rightarrow C : \{ack_2\}_{KS_{CS}}$

Firstly, the client C sends its identifer *username* to the server S (**M1**). Receiving it, the server confirms *username* within its database. If there exists *username*, or even if not, the server notifies the acknowledgement ack_1 to prompt the next (**M2**). And then, the client sends its plaintext *password*, (encrypted with the session key) (**M3**). Finally, the server returns the acknowledgement ack_2, which denotes success or failure of the authentication (**M4**).

3.3.2 Public Key Authentication in SSH1

M1) $C \rightarrow S : \{username\}_{KS_{CS}}$
M2) $S \rightarrow C : \{ack_1\}_{KS_{CS}}$
M3) $C \rightarrow S : \{modulus\}_{KS_{CS}}$
M4) $S \rightarrow C : \{\{challenge\}_{P_C}\}_{KS_{CS}}$
M5) $C \rightarrow S : \{f(challenge, S_{CS})\}_{KS_{CS}}$
M6) $S \rightarrow C : \{ack_2\}_{KS_{CS}}$

Firstly, same as **M1**, **M2** in SSH1 password authentication, the client C sends its identifer *username* to the server S (**M1**). Receiving it, the server confirms *username* within its database. If there exists *username*, or even if not, the server notifies the acknowledgement ack_1 to prompt the next (**M2**). Moreover, the client sends its public key's identifer *modulus* (**M3**). Receiving it, the server searches its public key according to *modulus* within its database. It also creates *challenge*, encrypts it with the public key P_C, returns it to the client (**M4**). And then, the client decrypts the encrypted message with its private key P_C^{-1}, computes the MD5 hash value of concatenation of *challenge* and S_{CS}, and returns the hash value to the server (**M5**). Finally, the server verifies the response, returns the acknowledgement ack_2, which denotes success or failure of the authentication (**M6**).

3.3.3 Password Authentication in SSH2

M1) $C \rightarrow S : \{username, password\}_{KS_{CS}}$
M2) $S \rightarrow C : \{ack\}_{KS_{CS}}$

Firstly, the client C sends its identifer *username* and plaintext *password* to the server S (**M1**). Note that *password* is also encrypted. Receiving it, the server confirms *username* and *password* within its database. Finally, the server returns the acknowledgement *ack*, which denotes success or failure of the authentication (**M2**).

3.3.4 Public Key Authentication in SSH2

M1) $C \rightarrow S :$
$\{username, P_C, \{username, P_C, S_{CS}\}_{P_C^{-1}}\}_{KS_{CS}}$
M2) $S \rightarrow C : \{ack\}_{KS_{CS}}$

Firstly, the client C sends its identifer *username*, its public key P_C, and the message $\{username, P_C, S_{CS}\}$ signed with its private key P_C^{-1}, to the server S (**M1**). Receiving it, the server confirms *username* and P_C within its database, and verifies the signature. Finally, the server returns the acknowledgement *ack*, which denotes success or failure of the authentication (**M2**).

4. Architectural Defects of the SSH

In this section, architectural safety of the SSH1 and SSH2 are considered. It is explained that there exists critical defects in the design of SSH1 and SSH2.

4.1 Security Considerations

After the key exchange phase, the SSH server and client share the session key, and the client identify the server by the authentication. This kind of authentication is same as that of the *server-authentication mode* in the SSL (Secure Socket Layer) protocol [12]. Which means that, the server can't prove to identify the client in this phase of SSH protocol.

According to *the requirement of the secure authentication with exchanging the session key* [11], since the server and client can't obtain the agreement of sharing it in secure way, the method of key-exchange in this phase is not secure to establish the communication channel between the server and client. Therefore, the server can be deceived by the intruder in MITM (man in the middle) attack: when the client connects with the intruder, which masquerades as the mirror server of the authorized one, the intruder deceives the authorized server. This kind of MITM attack is notorious for being shown in G.Lowe's paper [10].

Since the proper mutual authentication is not executed until the user authentication phase, there needs strong mutual authentication for the secure channel. Therefore, the naive or careless user can be deprived of its password. In the following section, the example of attack is provided.

4.2 Attack over the SSH1 protocols

For this attack, as it was said, there needs the assumption: the client firstly connects with the intruder masquerading the server. You may think, that's curious! However, if it is announced as the new mirror server, which can provide file resources shared by NFS (network file system), some users may believe it as the authorized mirror server and connect it. And most important thing is that the intruder does not

have any resource which the proper server has, and, needless to say, the intruder does not know the user's password.

4.2.1 Key Exchange Phase

M1)	$S \rightarrow I(C)$	:	P_S, H_S, SA, RN
M1')	$I \rightarrow C$	:	P_I, H_I, SA, RN
M2)	$C \rightarrow I$	:	$SA, RN, \{\{KS_{CI}\}_{P_I}\}_{H_I}$
M2')	$I(C) \rightarrow S$	:	$SA, RN, \{\{KS_{IS}\}_{P_S}\}_{H_S}$
M3)	$S \rightarrow I(C)$	:	$\{ack\}_{KS_{IS}}$
M3')	$I \rightarrow C$	:	$\{ack\}_{KS_{CI}}$

After the client connects with the intruder on the assumption, the server S sends the first messages to the intruder pretending the client C (**M1**). Receiving them, the intruder sends its original messages to the client (**M1'**). Then the client creates the session key KS_{CI} between the client C and intruder I, encrypts it, and returns it to the intruder I (**M2**). Receiving it, the intruder also creates the session key KS_{IS} between the intruder I and server S, encrypts it, and transmits it to the server S (**M2'**). And then, by receiving an acknowledgement encrypted with the KS_{IS} (**M3**), the intruder sends an acknowledgement encrypted with the KS_{CI}. The client can decide that the session key is shared properly with I (**M3'**).

Until now, the session identifer S_{CI} and session key KS_{CI} are shared between the client C and the intruder I, and also, S_{IS} and KS_{IS} are shared between I and S, The server S doesn't care about this situation, because, in this phase, there doesn't authenticate the client or user. Let's assume this situation, and go next.

4.2.2 User Authentication Phase

Password Authentication in SSH1

M1)	$C \rightarrow I$	:	$\{username_C\}_{KS_{CI}}$
M1')	$I(C) \rightarrow S$	:	$\{username_C\}_{KS_{IS}}$
M2)	$S \rightarrow I(C)$	:	$\{ack_1\}_{KS_{IS}}$
M2')	$I \rightarrow C$	:	$\{ack_1\}_{KS_{CI}}$
M3)	$C \rightarrow I$	:	$\{password_C\}_{KS_{CI}}$
M3')	$I(C) \rightarrow S$	:	$\{password_C\}_{KS_{IS}}$
M4)	$S \rightarrow I(C)$	:	$\{ack_2\}_{KS_{IS}}$
M4')	$I \rightarrow C$	:	$\{ack_2\}_{KS_{CI}}$

In this user authentication phase after the previous phase, the client C sends its identifer $username_C$ encrypted with KS_{CI}, to the intruder I (**M1**). No caring about the identifer, the intruder sends $username_C$ encrypted with KS_{IS} (**M1'**). Receiving it, the server confirms $username_C$ within its database. The server notifies the acknowledgement ack_1 to prompt the next (**M2**). The intruder decrypts it, sends ack_1 encrypted with KS_{CI} (**M2'**). And then, the client sends its plaintext password encrypted with KS_{CI} (**M3**). The intruder gets its password dexterously. Although it is unnecessary now, let's continue it for masquerading the mirror server. It also sends its password encrypted with KS_{IS} (**M3'**). Finally, the server returns the acknowledgement ack_2 (**M4**). The intruder also sends the acknowledgement (**M4'**). Moreover, the intruder can provide the client to read or write any files it wants.

You may think that, owing to foolishly connecting with the intruder, the user is not clever. Well, let's go to see the way of the public key authentication.

Public Key Authentication in SSH1

M1)	$C \rightarrow I$	:	$\{username_C\}_{KS_{CI}}$
M1')	$I(C) \rightarrow S$	:	$\{username_C\}_{KS_{IS}}$
M2)	$S \rightarrow I(C)$	:	$\{ack_1\}_{KS_{IS}}$
M2')	$I \rightarrow C$	:	$\{ack_1\}_{KS_{CI}}$
M3)	$C \rightarrow I$	:	$\{modulus_C\}_{KS_{CI}}$
M3')	$I(C) \rightarrow S$	:	$\{modulus_C\}_{KS_{IS}}$
M4)	$S \rightarrow I(C)$	:	$\{\{challenge\}_{P_C}\}_{KS_{IS}}$
M4')	$I \rightarrow C$	:	$\{\{challenge\}_{P_C}\}_{KS_{CI}}$
M5)	$C \rightarrow I$	:	$\{f(challenge, S_{CI})\}_{KS_{CI}}$

Firstly, the client C sends its identifer $username_C$ to the intruder I (**M1**). And then, the intruder I pretending the client C sends the identifer to the server S (**M1'**). Receiving it, the server confirms $username_C$ within its database. The server notifies the acknowledgement ack_1 to prompt the next (**M2**). The intruder decrypts it, sends ack_1 encrypted with KS_{CI} (**M2'**). And then, the client sends *modulus* encrypted with KS_{CI} (**M3**). It also decrypt it, and sends it encrypted with KS_{IS} (**M3'**). Receiving it, the server creates *challenge*, encrypts it with the public key P_C, returns it to the intruder pretending the client (**M4**). Note that, since *challenge* is encrypted with the public key P_C, the intruder can't decrypt to obtain *challenge*. Therefore, the intruder sends $\{challenge\}_{P_C}$ to the client (**M4'**). And then, the client decrypts the encrypted message with its private key P_C^{-1}, computes the hash value of concatenation of *challenge* and S_{CI}, and returns the hash value (**M5**). Well, since the intruder and server shares the session identifer S_{IS}, the intruder should make the message such as $\{f(challenge, S_{IS})\}_{KS_{IS}}$. However, since the intruder doesn't know *challenge*, it can't make the message! Therefore, the SSH1 protocols with public key defeat the intruder!! If the intruder continue the session, the server can detect its existence.

4.3 Attack over the SSH2 protocols

As well as SSH1, there needs the assumption: the client

firstly connects with the intruder masquerading the server, e.g. the mirror server.

4.3.1 Key Exchange Phase

M1) $S \leftrightarrow I(C)$: SA
M1′) $I \leftrightarrow C$: SA
M2) $C \rightarrow I$: K_C
M2′) $I(C) \rightarrow S$: K_I
M3) $S \rightarrow I(C)$: $K_S, P_S, \{f(V_I, V_S, I_I, I_S, P_S, K_I, K_S, KS_{IS})\}_{P_S^{-1}}$
M3′) $I \rightarrow C$: $K_I, P_I, \{f(V_C, V_I, I_C, I_I, P_I, K_C, K_I, KS_{CI})\}_{P_I^{-1}}$
M4) $C \rightarrow I$: $\{KS_{CI}\}_{KS_{CI}}$
M4′) $I(C) \rightarrow S$: $\{KS_{IS}\}_{KS_{IS}}$

Firstly, the client C and intruder I negotiate some session parameters with SA (**M1**), similarly, the intruder pretending the client and server S negotiate (**M1′**). Next, the client C computes K_C, sends it to the intruder I (**M2**), and the intruder also computes K_I, and sends it to the server S (**M2′**). After receiving it, the server S computes the session key KS_{IS}. And, it also computes H, which is the session identifer, a hash value of a concatenation of V_I, V_S, I_I, I_S, P_S, K_I, K_S, and KS_{IS}. Moreover, the server sends K_S, the host key P_S, and the hash value H signed with P_S^{-1} (**M3**). And also, the intruder computes the values, and sends K_I, the host key P_I, and the hash value signed with its private host key P_I^{-1} (**M3′**). When the client receives them, using by its database, it confirm if P_I^{-1} really is the host key correspoding to I. and then, the client also computes the session key KS_{CI}, and verifies the signature of H with its host key P_I. Finally, the client and server confirms the session key is shared (**M4** and **M4′**).

Now, the session identifer S_{CI} and session key KS_{CI} are shared between the client C and the intruder I, and also, S_{IS} and KS_{IS} are shared between I and S.

4.3.2 User Authentication Phase

Password Authentication in SSH2

M1) $C \rightarrow I$: $\{username_C, password_C\}_{KS_{CI}}$
M1′) $I(C) \rightarrow S$: $\{username_C, password_C\}_{KS_{IS}}$
M2) $S \rightarrow I(C)$: $\{ack\}_{KS_{IS}}$
M2′) $I \rightarrow C$: $\{ack\}_{KS_{CI}}$

In this user authentication phase, the client C sends $username_C$ and $password_C$ encrypted with KS_{CI}, to the intruder I (**M1**). The intruder obtains its password in this step. For masquerading the mirror server, it continues the session. The intruder I sends $username_C$ and $password_C$ to the server S (**M1′**). Receiving it, the server confirms $username_C$ and $password_C$ within its database. Finally, the server returns the acknowledgement ack, which denotes success or failure of the authentication (**M2**). And also, the intruder notifies the acknowledgement (**M2′**).

Public Key Authentication in SSH2

M1) $C \rightarrow I$:
$\{username_C, P_C, \{username_C, P_C, S_{CI}\}_{P_C^{-1}}\}_{KS_{CI}}$

The client C sends its identifer $username_C$, its public key P_C, and the message $\{username_C, P_C, S_{CI}\}$ signed with its private host key P_C^{-1}, to the intruder I (**M1**). For masquerading the client C, in the next step, the intruder I should make the message $\{username_C, P_C, S_{IS}\}_{P_C^{-1}}$. Since the intruder does not have the C's private key P_C^{-1}, it can make the message. Note that the session identifer S_{IS} in this massage is different from one in the C's message. The SSH2 protocols with public key prevent the intruder's attack.

5. Discussions

5.1 For Improvement

As it was explained in the section 3.1, the intruder masquerading the server can compel the client to decide the way of authentication proposed by the intruder. Therefore, as the password authentication holds defects, the intruder can force to utilize the password way if the user continue the session. On one side, user authentication with the public key can prevent the above intruder, on the other side, user authentication with password can't prevent it. This inconsistency obviously is caused by the defect of the SSH.

Well, to prevent the attack on password authentication, for example, one of the following changes is essential in the password authentication phase:

(1) The plain text password must not be transmitted. Naturally, the session key and the server's public can't be trusted.

(2) The server utilizes the challenge and response scheme: after the server sends the random challenge encrypted with the password, it can identify the client if receiving the proper modified response.

5.2 Related Work

There is the *dsniff* package [13], which can execute the MITM attack to the SSH1 protocols. This pack includes the programs `sshmitm` and `dnsspoof`. The former can snatch the session from the client C to the server S, and transmit its public key for *pretending the server S*. The latter can rewrite the entry in the DNS (domain name system) server.

The following is the dsniff's attack scenario:

M0) $C \rightarrow I(S)$: $request_C$

M0') $I(C) \rightarrow S$: $request_C$

M1) $S \rightarrow I(C)$: P_S, H_S, SA, RN

M1') $I(S) \rightarrow C$: P_I, H_I, SA, RN

M2) $C \rightarrow I(S)$: $SA, RN, \{\{KS_{CI}\}_{P_I}\}_{H_I}$

M2') $I(C) \rightarrow S$: $SA, RN, \{\{KS_{IS}\}_{P_S}\}_{H_S}$

M3) $S \rightarrow I(C)$: $\{ack\}_{KS_{IS}}$

M3') $I(S) \rightarrow C$: $\{ack\}_{KS_{CI}}$

Firstly, when the client C connects to the server S, using `dnsspoof`, the intruder deprives the session to connect with `sshmitm` on its machine (**M0** and **M0'**). Nextly, the server sends its reply (**M1**), and the intruder also sends the corresponding messages (**M1'**). In this step, importantly, since the client C intends to the server S, it thinks that the host key must be P_S. However the received host key P_I is different from the key in its database, so the client program warns the change. A nervous or careful user may perceive danger, but someone may not. Moreover, the SSH design may not guarantee to protect such optimistic user.

Well, we confirm our attack over SSH1 can be executed by using this tool. However, there is a big difference: the intruder assumed in the `dnsspoof` pretends the server. Since this tool is designed for the session hijack attack to pretend the server S, it transmits S' environment parameters, e.g. the server's identifer. So, there needs to modify the tool for our attack.

6. Conclusion

In this paper, we consider the architecture of the SSH, provide the defects of the SSH in the password user authentication after the key exchange phase. And more, owing to the SSH design, the user using the public key authentication may be force to expose its password if it wants to continue the session. Possibly the SSH designers may predict this attack and say no problem. However, it must cause a kind of damage to its user, and this is the defect which can be avoided in its design.

文　献

[1] Daniel J. Barrett, Richard E. Silverman: SSH, The Secure Shell, O'REILLY, February 2001, ISBN:0-596-00011-1.

[2] T.Ylonen: The SSH (Secure Shell) Remote Login Protocol, Internet Draft,Network Working Group, November 1995.

[3] T.Ylonen, T.Kivinen, M.Saarinen, T.Rinne, and S.Lehtinen: SSH Protocol Architecture `draft-ietf-secsh-architecture-11.txt`, Internet Draft, Network Working Group, November 2001.

[4] T.Ylonen, T.Kivinen, M.Saarinen, T.Rinne, and S.Lehtinen: SSH Transport Layer Protocol `draft-ietf-secsh-transport-11.txt`, Internet Draft, Network Working Group, November 2001.

[5] T.Ylonen, T.Kivinen, M.Saarinen, T.Rinne, and S.Lehtinen: SSH Connection Protocol `draft-ietf-secsh-connect-14.txt`, Internet Draft,Network Working Group, November 2001.

[6] T.Ylonen, T.Kivinen, M.Saarinen, T.Rinne, and S.Lehtinen: SSH Authentication Protocol `draft-ietf-secsh-userauth-13.txt`, Internet Draft,Network Working Group, November 2001.

[7] M.Friedl, N.Probos, W.Simpsom: Diffie-Hellman Group Exchange for the SSH Transport Layer Protocol `draft-ietf-secsh-dh-group-exchange-00.txt`, INTERNET DRAFT, Network Working Group, April 2001.

[8] `http://www.ssh.com/`

[9] `http://www.openssh.com/index.html`

[10] G.Lowe : Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR, Tools and Algorithms for Construction and Analysis of Systems, pp147-166, 1996.

[11] T.Saito, M.Hagiya, F.Mizoguchi: On Authentication Protocols Using Public-Key Cryptography, IPSJ (Information Processing Society of Japan) Journal, Vol.42, No8, pp2040-2048 (in Japanese), 2001.

[12] A.Freier, P.Kaltorn, and P.Kocher : The SSL Protocol Version 3.0, `http://home.netscape.com/eng/ssl3/draft302.txt`

[13] `http://www.monkey.org/ dugsong/dsniff/`