Network Working Group
Internet-Draft
Expires: March 31, 2004

T. Ylonen
SSH Communications Security Corp
D. Moffat, Ed.
Sun Microsystems, Inc
Oct 2003

SSH Protocol Architecture
draft-ietf-secsh-architecture-15.txt

Status of this Memo

This Internet-Draft will expire on March 31, 2004.

Copyright Notice

Abstract

SSH is a protocol for secure remote login and other secure network services over an insecure network. This document describes the architecture of the SSH protocol, as well as the notation and terminology used in SSH protocol documents. It also discusses the SSH algorithm naming system that allows local extensions. The SSH protocol consists of three major components: The Transport Layer Protocol provides server authentication, confidentiality, and integrity with perfect forward secrecy. The User Authentication Protocol authenticates the client to the server. The Connection Protocol multiplexes the encrypted tunnel into several logical channels. Details of these protocols are described in separate documents.

Table of Contents

1. Contributors

The major original contributors of this document were: Tatu Ylonen, Tero Kivinen, Timo J. Rinne, Sami Lehtinen (all of SSH Communications Security Corp), and Markku-Juhani O. Saarinen (University of Jyvaskyla)

The document editor is: Darren.Moffat@Sun.COM. Comments on this internet draft should be sent to the IETF SECSH working group, details at: http://ietf.org/html.charters/secsh-charter.html

2. Introduction

SSH is a protocol for secure remote login and other secure network services over an insecure network. It consists of three major components:

- The Transport Layer Protocol [SSH-TRANS] provides server authentication, confidentiality, and integrity. It may optionally also provide compression. The transport layer will typically be run over a TCP/IP connection, but might also be used on top of any other reliable data stream.
- The User Authentication Protocol [SSH-USERAUTH] authenticates the client-side user to the server. It runs over the transport layer protocol.
- The Connection Protocol [SSH-CONNECT] multiplexes the encrypted tunnel into several logical channels. It runs over the user authentication protocol.

The client sends a service request once a secure transport layer connection has been established. A second service request is sent after user authentication is complete. This allows new protocols to be defined and coexist with the protocols listed above.

The connection protocol provides channels that can be used for a wide range of purposes. Standard methods are provided for setting up secure interactive shell sessions and for forwarding ("tunneling") arbitrary TCP/IP ports and X11 connections.

3. Specification of Requirements

All documents related to the SSH protocols shall use the keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" to describe requirements. They are to be interpreted as described in [RFC2119].

4. Architecture

4.1 Host Keys

Each server host SHOULD have a host key. Hosts MAY have multiple host keys using multiple different algorithms. Multiple hosts MAY share the same host key. If a host has keys at all, it MUST have at least one key using each REQUIRED public key algorithm (DSS [FIPS-186]).

The server host key is used during key exchange to verify that the client is really talking to the correct server. For this to be possible, the client must have a priori knowledge of the server's public host key.

Two different trust models can be used:

- The client has a local database that associates each host name (as typed by the user) with the corresponding public host key. This method requires no centrally administered infrastructure, and no third-party coordination. The downside is that the database of name-to-key associations may become burdensome to maintain.
- The host name-to-key association is certified by some trusted certification authority. The client only knows the CA root key, and can verify the validity of all host keys certified by accepted CAs.

  The second alternative eases the maintenance problem, since ideally only a single CA key needs to be securely stored on the client. On the other hand, each host key must be appropriately certified by a central authority before authorization is possible. Also, a lot of trust is placed on the central infrastructure.

The protocol provides the option that the server name - host key association is not checked when connecting to the host for the first time. This allows communication without prior communication of host keys or certification. The connection still provides protection against passive listening; however, it becomes vulnerable to active man-in-the-middle attacks. Implementations SHOULD NOT normally allow such connections by default, as they pose a potential security problem. However, as there is no widely deployed key infrastructure available on the Internet yet, this option makes the protocol much more usable during the transition time until such an infrastructure emerges, while still providing a much higher level of security than that offered by older solutions (e.g. telnet [RFC-854] and rlogin [RFC-1282]).

Implementations SHOULD try to make the best effort to check host keys. An example of a possible strategy is to only accept a host key without checking the first time a host is connected, save the key in a local database, and compare against that key on all future connections to that host.

Implementations MAY provide additional methods for verifying the correctness of host keys, e.g. a hexadecimal fingerprint derived from the SHA-1 hash of the public key. Such fingerprints can easily be verified by using telephone or other external communication channels.

All implementations SHOULD provide an option to not accept host keys that cannot be verified.

We believe that ease of use is critical to end-user acceptance of security solutions, and no improvement in security is gained if the new solutions are not used. Thus, providing the option not to check the server host key is believed to improve the overall security of the Internet, even though it reduces the security of the protocol in configurations where it is allowed.

4.2 Extensibility

We believe that the protocol will evolve over time, and some organizations will want to use their own encryption, authentication and/or key exchange methods. Central registration of all extensions is cumbersome, especially for experimental or classified features. On the other hand, having no central registration leads to conflicts in method identifiers, making interoperability difficult.

We have chosen to identify algorithms, methods, formats, and extension protocols with textual names that are of a specific format. DNS names are used to create local namespaces where experimental or classified extensions can be defined without fear of conflicts with other implementations.

One design goal has been to keep the base protocol as simple as possible, and to require as few algorithms as possible. However, all implementations MUST support a minimal set of algorithms to ensure interoperability (this does not imply that the local policy on all hosts would necessary allow these algorithms). The mandatory algorithms are specified in the relevant protocol documents.

Additional algorithms, methods, formats, and extension protocols can be defined in separate drafts. See Section Algorithm Naming (Section 6) for more information.

4.3 Policy Issues

The protocol allows full negotiation of encryption, integrity, key exchange, compression, and public key algorithms and formats. Encryption, integrity, public key, and compression algorithms can be

different for each direction.

The following policy issues SHOULD be addressed in the configuration mechanisms of each implementation:
- Encryption, integrity, and compression algorithms, separately for each direction. The policy MUST specify which is the preferred algorithm (e.g. the first algorithm listed in each category).
- Public key algorithms and key exchange method to be used for host authentication. The existence of trusted host keys for different public key algorithms also affects this choice.
- The authentication methods that are to be required by the server for each user. The server's policy MAY require multiple authentication for some or all users. The required algorithms MAY depend on the location where the user is trying to log in from.
- The operations that the user is allowed to perform using the connection protocol. Some issues are related to security; for example, the policy SHOULD NOT allow the server to start sessions or run commands on the client machine, and MUST NOT allow connections to the authentication agent unless forwarding such connections has been requested. Other issues, such as which TCP/IP ports can be forwarded and by whom, are clearly issues of local policy. Many of these issues may involve traversing or bypassing firewalls, and are interrelated with the local security policy.

4.4 Security Properties

The primary goal of the SSH protocol is improved security on the Internet. It attempts to do this in a way that is easy to deploy, even at the cost of absolute security.
- All encryption, integrity, and public key algorithms used are well-known, well-established algorithms.
- All algorithms are used with cryptographically sound key sizes that are believed to provide protection against even the strongest cryptanalytic attacks for decades.
- All algorithms are negotiated, and in case some algorithm is broken, it is easy to switch to some other algorithm without modifying the base protocol.

Specific concessions were made to make wide-spread fast deployment easier. The particular case where this comes up is verifying that the server host key really belongs to the desired host; the protocol allows the verification to be left out (but this is NOT RECOMMENDED). This is believed to significantly improve usability in the short term, until widespread Internet public key infrastructures emerge.

4.5 Packet Size and Overhead

Some readers will worry about the increase in packet size due to new headers, padding, and MAC. The minimum packet size is in the order of 28 bytes (depending on negotiated algorithms). The increase is negligible for large packets, but very significant for one-byte packets (telnet-type sessions). There are, however, several factors that make this a non-issue in almost all cases:

- The minimum size of a TCP/IP header is 32 bytes. Thus, the increase is actually from 33 to 51 bytes (roughly).
- The minimum size of the data field of an Ethernet packet is 46 bytes [RFC-894]. Thus, the increase is no more than 5 bytes. When Ethernet headers are considered, the increase is less than 10 percent.
- The total fraction of telnet-type data in the Internet is negligible, even with increased packet sizes.

The only environment where the packet size increase is likely to have a significant effect is PPP [RFC-1134] over slow modem lines (PPP compresses the TCP/IP headers, emphasizing the increase in packet size). However, with modern modems, the time needed to transfer is in the order of 2 milliseconds, which is a lot faster than people can type.

There are also issues related to the maximum packet size. To minimize delays in screen updates, one does not want excessively large packets for interactive sessions. The maximum packet size is negotiated separately for each channel.

4.6 Localization and Character Set Support

For the most part, the SSH protocols do not directly pass text that would be displayed to the user. However, there are some places where such data might be passed. When applicable, the character set for the data MUST be explicitly specified. In most places, ISO 10646 with UTF-8 encoding is used [RFC-2279]. When applicable, a field is also provided for a language tag [RFC-3066].

One big issue is the character set of the interactive session. There is no clear solution, as different applications may display data in different formats. Different types of terminal emulation may also be employed in the client, and the character set to be used is effectively determined by the terminal emulation. Thus, no place is provided for directly specifying the character set or encoding for terminal session data. However, the terminal emulation type (e.g. "vt100") is transmitted to the remote site, and it implicitly specifies the character set and encoding. Applications typically use the terminal type to determine what character set they use, or the character set is determined using some external means. The terminal emulation may also allow configuring the default character set. In any case, the character set for the terminal session is considered primarily a client local issue.

Internal names used to identify algorithms or protocols are normally never displayed to users, and must be in US-ASCII.

The client and server user names are inherently constrained by what the server is prepared to accept. They might, however, occasionally be displayed in logs, reports, etc. They MUST be encoded using ISO 10646 UTF-8, but other encodings may be required in some cases. It is up to the server to decide how to map user names to accepted user names. Straight bit-wise binary comparison is RECOMMENDED.

For localization purposes, the protocol attempts to minimize the number of textual messages transmitted. When present, such messages typically relate to errors, debugging information, or some externally configured data. For data that is normally displayed, it SHOULD be possible to fetch a localized message instead of the transmitted message by using a numerical code. The remaining messages SHOULD be configurable.

5. Data Type Representations Used in the SSH Protocols

byte

A byte represents an arbitrary 8-bit value (octet) [RFC-1700]. Fixed length data is sometimes represented as an array of bytes, written byte[n], where n is the number of bytes in the array.

boolean

A boolean value is stored as a single byte. The value 0 represents FALSE, and the value 1 represents TRUE. All non-zero values MUST be interpreted as TRUE; however, applications MUST NOT store values other than 0 and 1.

uint32

Represents a 32-bit unsigned integer. Stored as four bytes in the order of decreasing significance (network byte order). For example, the value 699921578 (0x29b7f4aa) is stored as 29 b7 f4 aa.

uint64

Represents a 64-bit unsigned integer. Stored as eight bytes in the order of decreasing significance (network byte order).

string

Arbitrary length binary string. Strings are allowed to contain arbitrary binary data, including null characters and 8-bit characters. They are stored as a uint32 containing its length (number of bytes that follow) and zero (= empty string) or more bytes that are the value of the string. Terminating null characters are not used.

Strings are also used to store text. In that case, US-ASCII is used for internal names, and ISO-10646 UTF-8 for text that might be displayed to the user. The terminating null character SHOULD NOT normally be stored in the string.

For example, the US-ASCII string "testing" is represented as 00 00 00 07 t e s t i n g. The UTF8 mapping does not alter the encoding of US-ASCII characters.

mpint

Represents multiple precision integers in two’s complement format, stored as a string, 8 bits per byte, MSB first. Negative numbers have the value 1 as the most significant bit of the first byte of the data partition. If the most significant bit would be set for a positive number, the number MUST be preceded by a zero byte. Unnecessary leading bytes with the value 0 or 255 MUST NOT be included. The value zero MUST be stored as a string with zero bytes of data.

By convention, a number that is used in modular computations in Z_n SHOULD be represented in the range 0 <= x < n.

Examples:

value (hex)	representation (hex)
0	00 00 00 00
9a378f9b2e332a7	00 00 00 08 09 a3 78 f9 b2 e3 32 a7
80	00 00 00 02 00 80
-1234	00 00 00 02 ed cc
-deadbeef	00 00 00 05 ff 21 52 41 11

name-list

A string containing a comma separated list of names. A name list is represented as a uint32 containing its length (number of bytes that follow) followed by a comma-separated list of zero or more

names. A name MUST be non-zero length, and it MUST NOT contain a comma (’,’). Context may impose additional restrictions on the names; for example, the names in a list may have to be valid algorithm identifier (see Algorithm Naming below), or [RFC-3066] language tags. The order of the names in a list may or may not be significant, also depending on the context where the list is is used. Terminating NUL characters are not used, neither for the individual names, nor for the list as a whole.

Examples:

value	representation (hex)
(), the empty list	00 00 00 00
("zlib")	00 00 00 04 7a 6c 69 62
("zlib", "none")	00 00 00 09 7a 6c 69 62 2c 6e 6f 6e 65

6. Algorithm Naming

The SSH protocols refer to particular hash, encryption, integrity, compression, and key exchange algorithms or protocols by names. There are some standard algorithms that all implementations MUST support. There are also algorithms that are defined in the protocol specification but are OPTIONAL. Furthermore, it is expected that some organizations will want to use their own algorithms.

In this protocol, all algorithm identifiers MUST be printable US-ASCII non-empty strings no longer than 64 characters. Names MUST be case-sensitive.

There are two formats for algorithm names:

- o Names that do not contain an at-sign (@) are reserved to be assigned by IETF consensus (RFCs). Examples include ‘3des-cbc’, ‘sha-1’, ‘hmac-sha1’, and ‘zlib’ (the quotes are not part of the name). Names of this format MUST NOT be used without first registering them. Registered names MUST NOT contain an at-sign (@) or a comma (,).
- o Anyone can define additional algorithms by using names in the format name@domainname, e.g. "ourcipher-cbc@example.com". The format of the part preceding the at sign is not specified; it MUST consist of US-ASCII characters except at-sign and comma. The part following the at-sign MUST be a valid fully qualified internet domain name [RFC-1034] controlled by the person or organization defining the name. It is up to each domain how it manages its local namespace.

7. Message Numbers

SSH packets have message numbers in the range 1 to 255. These numbers have been allocated as follows:

Transport layer protocol:

1 to 19	Transport layer generic (e.g. disconnect, ignore, debug, etc.)
20 to 29	Algorithm negotiation
30 to 49	Key exchange method specific (numbers can be reused for different authentication methods)

User authentication protocol:

50 to 59	User authentication generic
60 to 79	User authentication method specific (numbers can be reused for different authentication methods)

Connection protocol:

80 to 89	Connection protocol generic
90 to 127	Channel related messages

Reserved for client protocols:

128 to 191	Reserved

Local extensions:

192 to 255	Local extensions

8. IANA Considerations

The initial state of the IANA registry is detailed in [SSH-NUMBERS].

Allocation of the following types of names in the SSH protocols is assigned by IETF consensus:

- o SSH encryption algorithm names,
- o SSH MAC algorithm names,
- o SSH public key algorithm names (public key algorithm also implies encoding and signature/encryption capability),
- o SSH key exchange method names, and
- o SSH protocol (service) names.

These names MUST be printable US-ASCII strings, and MUST NOT contain the characters at-sign ('@'), comma (','), or whitespace or control characters (ASCII codes 32 or less). Names are case-sensitive, and MUST NOT be longer than 64 characters.

Names with the at-sign ('@') in them are allocated by the owner of DNS name after the at-sign (hierarchical allocation in [RFC-2343]), otherwise the same restrictions as above.

Each category of names listed above has a separate namespace. However, using the same name in multiple categories SHOULD be avoided to minimize confusion.

Message numbers (see Section Message Numbers (Section 7)) in the range of 0..191 are allocated via IETF consensus; message numbers in the 192..255 range (the "Local extensions" set) are reserved for private use.

9. Security Considerations

In order to make the entire body of Security Considerations more accessible, Security Considerations for the transport, authentication, and connection documents have been gathered here.

The transport protocol [1] provides a confidential channel over an insecure network. It performs server host authentication, key exchange, encryption, and integrity protection. It also derives a unique session id that may be used by higher-level protocols.

The authentication protocol [2] provides a suite of mechanisms which can be used to authenticate the client user to the server. Individual mechanisms specified in the in authentication protocol use the session id provided by the transport protocol and/or depend on the security and integrity guarantees of the transport protocol.

The connection protocol [3] specifies a mechanism to multiplex multiple streams [channels] of data over the confidential and authenticated transport. It also specifies channels for accessing an interactive shell, for 'proxy-forwarding' various external protocols over the secure transport (including arbitrary TCP/IP protocols), and for accessing secure 'subsystems' on the server host.

9.1 Pseudo-Random Number Generation

This protocol binds each session key to the session by including random, session specific data in the hash used to produce session keys. Special care should be taken to ensure that all of the random numbers are of good quality. If the random data here (e.g., DH parameters) are pseudo-random then the pseudo-random number generator should be cryptographically secure (i.e., its next output not easily guessed even when knowing all previous outputs) and, furthermore, proper entropy needs to be added to the pseudo-random number generator. RFC 1750 [1750] offers suggestions for sources of random numbers and entropy. Implementors should note the importance of entropy and the well-meant, anecdotal warning about the difficulty in properly implementing pseudo-random number generating functions.

The amount of entropy available to a given client or server may sometimes be less than what is required. In this case one must either resort to pseudo-random number generation regardless of insufficient entropy or refuse to run the protocol. The latter is preferable.

9.2 Transport

9.2.1 Confidentiality

It is beyond the scope of this document and the Secure Shell Working Group to analyze or recommend specific ciphers other than the ones which have been established and accepted within the industry. At the time of this writing, ciphers commonly in use include 3DES, ARCFOUR, twofish, serpent and blowfish. AES has been accepted by The published as a US Federal Information Processing Standards [FIPS-197] and the cryptographic community as being acceptable for this purpose as well has accepted AES. As always, implementors and users should check current literature to ensure that no recent vulnerabilities have been found in ciphers used within products. Implementors should also check to see which ciphers are considered to be relatively stronger than others and should recommend their use to users over relatively weaker ciphers. It would be considered good form for an implementation to politely and unobtrusively notify a user that a stronger cipher is available and should be used when a weaker one is actively chosen.

The "none" cipher is provided for debugging and SHOULD NOT be used except for that purpose. It’s cryptographic properties are sufficiently described in RFC 2410, which will show that its use does not meet the intent of this protocol.

The relative merits of these and other ciphers may also be found in current literature. Two references that may provide information on the subject are [SCHNEIER] and [KAUFMAN,PERLMAN,SPECINER]. Both of these describe the CBC mode of operation of certain ciphers and the weakness of this scheme. Essentially, this mode is theoretically vulnerable to chosen cipher-text attacks because of the high predictability of the start of packet sequence. However, this attack

is still deemed difficult and not considered fully practicable especially if relatively longer block sizes are used.

Additionally, another CBC mode attack may be mitigated through the insertion of packets containing SSH_MSG_IGNORE. Without this technique, a specific attack may be successful. For this attack (commonly known as the Rogaway attack [ROGAWAY],[DAI],[BELLARE,KOHNO,NAMPREMPRE]) to work, the attacker would need to know the IV of the next block that is going to be encrypted. In CBC mode that is the output of the encryption of the previous block. If the attacker does not have any way to see the packet yet (i.e it is in the internal buffers of the ssh implementation or even in the kernel) then this attack will not work. If the last packet has been sent out to the network (i.e the attacker has access to it) then he can use the attack.

In the optimal case an implementor would need to add an extra packet only if the packet has been sent out onto the network and there are no other packets waiting for transmission. Implementors may wish to check to see if there are any unsent packets awaiting transmission, but unfortunately it is not normally easy to obtain this information from the kernel or buffers. If there are not, then a packet containing SSH_MSG_IGNORE SHOULD be sent. If a new packet is added to the stream every time the attacker knows the IV that is supposed to be used for the next packet, then the attacker will not be able to guess the correct IV, thus the attack will never be successfull.

As an example, consider the following case:

```
      Client                                            Server
      ------                                            ------
      TCP(seq=x, len=500)             ->
      contains Record 1

                          [500 ms passes, no ACK]

     TCP(seq=x, len=1000)             ->
      contains Records 1,2

                                                          ACK
```

1. The Nagle algorithm + TCP retransmits mean that the two records get coalesced into a single TCP segment
2. Record 2 is *not* at the beginning of the TCP segment and never will be, since it gets ACKed.

3. Yet, the attack is possible because Record 1 has already been seen.

As this example indicates, it's totally unsafe to use the existence of unflushed data in the TCP buffers proper as a guide to whether you need an empty packet, since when you do the second write(), the buffers will contain the un-ACKed Record 1.

On the other hand, it's perfectly safe to have the following situation:

```
     Client                                                Server
     ------                                                ------
     TCP(seq=x, len=500)           ->
        contains SSH_MSG_IGNORE

     TCP(seq=y, len=500)           ->
        contains Data
```

Provided that the IV for second SSH Record is fixed after the data for the Data packet is determined -i.e. you do:

```
     read from user
     encrypt null packet
     encrypt data packet
```

9.2.2 Data Integrity

This protocol does allow the Data Integrity mechanism to be disabled. Implementors SHOULD be wary of exposing this feature for any purpose other than debugging. Users and administrators SHOULD be explicitly warned anytime the "none" MAC is enabled.

So long as the "none" MAC is not used, this protocol provides data integrity.

Because MACs use a 32 bit sequence number, they might start to leak information after 2**32 packets have been sent. However, following the rekeying recommendations should prevent this attack. The transport protocol [1] recommends rekeying after one gigabyte of data, and the smallest possible packet is 16 bytes. Therefore, rekeying SHOULD happen after 2**28 packets at the very most.

9.2.3 Replay

The use of a MAC other than 'none' provides integrity and authentication. In addition, the transport protocol provides a unique session identifier (bound in part to pseudo-random data that is part of the algorithm and key exchange process) that can be used by higher level protocols to bind data to a given session and prevent replay of data from prior sessions. For example, the authentication protocol uses this to prevent replay of signatures from previous sessions. Because public key authentication exchanges are cryptographically bound to the session (i.e., to the initial key exchange) they cannot be successfully replayed in other sessions.

Note that the session ID can be made public without harming the security of the protocol.

If two session happen to have the same session ID [hash of key exchanges] then packets from one can be replayed against the other. It must be stressed that the chances of such an occurrence are, needless to say, minimal when using modern cryptographic methods. This is all the more so true when specifying larger hash function outputs and DH parameters.

Replay detection using monotonically increasing sequence numbers as input to the MAC, or HMAC in some cases, is described in [RFC2085] /> [RFC2246], [RFC2743], [RFC1964], [RFC2025], and [RFC1510]. The underlying construct is discussed in [RFC2104]. Essentially a different sequence number in each packet ensures that at least this one input to the MAC function will be unique and will provide a nonrecurring MAC output that is not predictable to an attacker. If the session stays active long enough, however, this sequence number will wrap. This event may provide an attacker an opportunity to replay a previously recorded packet with an identical sequence number but only if the peers have not rekeyed since the transmission of the first packet with that sequence number. If the peers have rekeyed, then the replay will be detected as the MAC check will fail. For this reason, it must be emphasized that peers MUST rekey before a wrap of the sequence numbers. Naturally, if an attacker does attempt to replay a captured packet before the peers have rekeyed, then the receiver of the duplicate packet will not be able to validate the MAC and it will be discarded. The reason that the MAC will fail is because the receiver will formulate a MAC based upon the packet contents, the shared secret, and the expected sequence number. Since the replayed packet will not be using that expected sequence number (the sequence number of the replayed packet will have already been passed by the receiver) then the calculated MAC will not match the MAC received with the packet.

9.2.4 Man-in-the-middle

This protocol makes no assumptions nor provisions for an infrastructure or means for distributing the public keys of hosts. It is expected that this protocol will sometimes be used without first verifying the association between the server host key and the server host name. Such usage is vulnerable to man-in-the-middle attacks. This section describes this and encourages administrators and users to understand the importance of verifying this association before any session is initiated.

There are three cases of man-in-the-middle attacks to consider. The first is where an attacker places a device between the client and the

server before the session is initiated. In this case, the attack device is trying to mimic the legitimate server and will offer its public key to the client when the client initiates a session. If it were to offer the public key of the server, then it would not be able to decrypt or sign the transmissions between the legitimate server and the client unless it also had access to the private-key of the host. The attack device will also, simultaneously to this, initiate a session to the legitimate server masquerading itself as the client. If the public key of the server had been securely distributed to the client prior to that session initiation, the key offered to the client by the attack device will not match the key stored on the client. In that case, the user SHOULD be given a warning that the offered host key does not match the host key cached on the client. As described in Section 3.1 of [ARCH], the user may be free to accept the new key and continue the session. It is RECOMMENDED that the warning provide sufficient information to the user of the client device so they may make an informed decision. If the user chooses to continue the session with the stored public-key of the server (not the public-key offered at the start of the session), then the session specific data between the attacker and server will be different between the client-to-attacker session and the attacker-to-server sessions due to the randomness discussed above. From this, the attacker will not be able to make this attack work since the attacker will not be able to correctly sign packets containing this session specific data from the server since he does not have the private key of that server.

The second case that should be considered is similar to the first case in that it also happens at the time of connection but this case points out the need for the secure distribution of server public keys. If the server public keys are not securely distributed then the client cannot know if it is talking to the intended server. An attacker may use social engineering techniques to pass off server keys to unsuspecting users and may then place a man-in-the-middle attack device between the legitimate server and the clients. If this is allowed to happen then the clients will form client-to-attacker sessions and the attacker will form attacker-to-server sessions and will be able to monitor and manipulate all of the traffic between the clients and the legitimate servers. Server administrators are encouraged to make host key fingerprints available for checking by some means whose security does not rely on the integrity of the actual host keys. Possible mechanisms are discussed in Section 3.1 of [SSH-ARCH] and may also include secured Web pages, physical pieces of paper, etc. Implementors SHOULD provide recommendations on how best to do this with their implementation. Because the protocol is extensible, future extensions to the protocol may provide better mechanisms for dealing with the need to know the server's host key before connecting. For example, making the host key fingerprint available through a secure DNS lookup, or using kerberos over gssapi during key exchange to authenticate the server are possibilities.

In the third man-in-the-middle case, attackers may attempt to manipulate packets in transit between peers after the session has been established. As described in the Replay part of this section, a successful attack of this nature is very improbable. As in the Replay section, this reasoning does assume that the MAC is secure and that it is infeasible to construct inputs to a MAC algorithm to give a known output. This is discussed in much greater detail in Section 6 of RFC 2104. If the MAC algorithm has a vulnerability or is weak enough, then the attacker may be able to specify certain inputs to yield a known MAC. With that they may be able to alter the contents of a packet in transit. Alternatively the attacker may be able to exploit the algorithm vulnerability or weakness to find the shared secret by reviewing the MACs from captured packets. In either of those cases, an attacker could construct a packet or packets that could be inserted into an SSH stream. To prevent that, implementors are encouraged to utilize commonly accepted MAC algorithms and administrators are encouraged to watch current literature and discussions of cryptography to ensure that they are not using a MAC algorithm that has a recently found vulnerability or weakness.

In summary, the use of this protocol without a reliable association of the binding between a host and its host keys is inherently insecure and is NOT RECOMMENDED. It may however be necessary in non-security critical environments, and will still provide protection against passive attacks. Implementors of protocols and applications running on top of this protocol should keep this possibility in mind.

9.2.5 Denial-of-service

This protocol is designed to be used over a reliable transport. If transmission errors or message manipulation occur, the connection is closed. The connection SHOULD be re-established if this occurs. Denial of service attacks of this type ("wire cutter") are almost impossible to avoid.

In addition, this protocol is vulnerable to Denial of Service attacks because an attacker can force the server to go through the CPU and memory intensive tasks of connection setup and key exchange without authenticating. Implementors SHOULD provide features that make this more difficult. For example, only allowing connections from a subset of IPs known to have valid users.

9.2.6 Covert Channels

The protocol was not designed to eliminate covert channels. For

example, the padding, SSH_MSG_IGNORE messages, and several other places in the protocol can be used to pass covert information, and the recipient has no reliable way to verify whether such information is being sent.

9.2.7 Forward Secrecy

It should be noted that the Diffie-Hellman key exchanges may provide perfect forward secrecy (PFS). PFS is essentially defined as the cryptographic property of a key-establishment protocol in which the compromise of a session key or long-term private key after a given session does not cause the compromise of any earlier session. [ANSI T1.523-2001] SSHv2 sessions resulting from a key exchange using diffie-hellman-group1-sha1 are secure even if private keying/ authentication material is later revealed, but not if the session keys are revealed. So, given this definition of PFS, SSHv2 does have PFS. It is hoped that all other key exchange mechanisms proposed and used in the future will also provide PFS. This property is not commuted to any of the applications or protocols using SSH as a transport however. The transport layer of SSH provides confidentiality for password authentication and other methods that rely on secret data.

Of course, if the DH private parameters for the client and server are revealed then the session key is revealed, but these items can be thrown away after the key exchange completes. It's worth pointing out that these items should not be allowed to end up on swap space and that they should be erased from memory as soon as the key exchange completes.

9.3 Authentication Protocol

The purpose of this protocol is to perform client user authentication. It assumes that this run over a secure transport layer protocol, which has already authenticated the server machine, established an encrypted communications channel, and computed a unique session identifier for this session.

Several authentication methods with different security characteristics are allowed. It is up to the server's local policy to decide which methods (or combinations of methods) it is willing to accept for each user. Authentication is no stronger than the weakest combination allowed.

The server may go into a "sleep" period after repeated unsuccessful authentication attempts to make key search more difficult for attackers. Care should be taken so that this doesn't become a self-denial of service vector.

9.3.1 Weak Transport

If the transport layer does not provide confidentiality, authentication methods that rely on secret data SHOULD be disabled. If it does not provide strong integrity protection, requests to change authentication data (e.g. a password change) SHOULD be disabled to prevent an attacker from modifying the ciphertext without being noticed, or rendering the new authentication data unusable (denial of service).

The assumption as stated above that the Authentication Protocol only run over a secure transport that has previously authenticated the server is very important to note. People deploying SSH are reminded of the consequences of man-in-the-middle attacks if the client does not have a very strong a priori association of the server with the host key of that server. Specifically for the case of the Authentication Protocol the client may form a session to a man-in-the-middle attack device and divulge user credentials such as their username and password. Even in the cases of authentication where no user credentials are divulged, an attacker may still gain information they shouldn't have by capturing key-strokes in much the same way that a honeypot works.

9.3.2 Debug messages

Special care should be taken when designing debug messages. These messages may reveal surprising amounts of information about the host if not properly designed. Debug messages can be disabled (during user authentication phase) if high security is required. Administrators of host machines should make all attempts to compartmentalize all event notification messages and protect them from unwarranted observation. Developers should be aware of the sensitive nature of some of the normal event messages and debug messages and may want to provide guidance to administrators on ways to keep this information away from unauthorized people. Developers should consider minimizing the amount of sensitive information obtainable by users during the authentication phase in accordance with the local policies. For this reason, it is RECOMMENDED that debug messages be initially disabled at the time of deployment and require an active decision by an administrator to allow them to be enabled. It is also RECOMMENDED that a message expressing this concern be presented to the administrator of a system when the action is taken to enable debugging messages.

9.3.3 Local security policy

Implementer MUST ensure that the credentials provided validate the professed user and also MUST ensure that the local policy of the

server permits the user the access requested. In particular, because of the flexible nature of the SSH connection protocol, it may not be possible to determine the local security policy, if any, that should apply at the time of authentication because the kind of service being requested is not clear at that instant. For example, local policy might allow a user to access files on the server, but not start an interactive shell. However, during the authentication protocol, it is not known whether the user will be accessing files or attempting to use an interactive shell, or even both. In any event, where local security policy for the server host exists, it MUST be applied and enforced correctly.

Implementors are encouraged to provide a default local policy and make its parameters known to administrators and users. At the discretion of the implementors, this default policy may be along the lines of 'anything goes' where there are no restrictions placed upon users, or it may be along the lines of 'excessively restrictive' in which case the administrators will have to actively make changes to this policy to meet their needs. Alternatively, it may be some attempt at providing something practical and immediately useful to the administrators of the system so they don't have to put in much effort to get SSH working. Whatever choice is made MUST be applied and enforced as required above.

9.3.4 Public key authentication

The use of public-key authentication assumes that the client host has not been compromised. It also assumes that the private-key of the server host has not been compromised.

This risk can be mitigated by the use of passphrases on private keys; however, this is not an enforceable policy. The use of smartcards, or other technology to make passphrases an enforceable policy is suggested.

The server could require both password and public-key authentication, however, this requires the client to expose its password to the server (see section on password authentication below.)

9.3.5 Password authentication

The password mechanism as specified in the authentication protocol assumes that the server has not been compromised. If the server has been compromised, using password authentication will reveal a valid username / password combination to the attacker, which may lead to further compromises.

This vulnerability can be mitigated by using an alternative form of authentication. For example, public-key authentication makes no assumptions about security on the server.

9.3.6 Host based authentication

Host based authentication assumes that the client has not been compromised. There are no mitigating strategies, other than to use host based authentication in combination with another authentication method.

9.4 Connection protocol

9.4.1 End point security

End point security is assumed by the connection protocol. If the server has been compromised, any terminal sessions, port forwarding, or systems accessed on the host are compromised. There are no mitigating factors for this.

If the client end point has been compromised, and the server fails to stop the attacker at the authentication protocol, all services exposed (either as subsystems or through forwarding) will be vulnerable to attack. Implementors SHOULD provide mechanisms for administrators to control which services are exposed to limit the vulnerability of other services.

These controls might include controlling which machines and ports can be target in 'port-forwarding' operations, which users are allowed to use interactive shell facilities, or which users are allowed to use exposed subsystems.

9.4.2 Proxy forwarding

The SSH connection protocol allows for proxy forwarding of other protocols such as SNMP, POP3, and HTTP. This may be a concern for network administrators who wish to control the access of certain applications by users located outside of their physical location. Essentially, the forwarding of these protocols may violate site specific security policies as they may be undetectably tunneled through a firewall. Implementors SHOULD provide an administrative mechanism to control the proxy forwarding functionality so that site specific security policies may be upheld.

In addition, a reverse proxy forwarding functionality is available, which again can be used to bypass firewall controls.

As indicated above, end-point security is assumed during proxy forwarding operations. Failure of end-point security will compromise

all data passed over proxy forwarding.

9.4.3 X11 forwarding

Another form of proxy forwarding provided by the ssh connection protocol is the forwarding of the X11 protocol. If end-point security has been compromised, X11 forwarding may allow attacks against the X11 server. Users and administrators should, as a matter of course, use appropriate X11 security mechanisms to prevent unauthorized use of the X11 server. Implementors, administrators and users who wish to further explore the security mechanisms of X11 are invited to read [SCHEIFLER] and analyze previously reported problems with the interactions between SSH forwarding and X11 in CERT vulnerabilities VU#363181 and VU#118892 [CERT].

X11 display forwarding with SSH, by itself, is not sufficient to correct well known problems with X11 security [VENEMA]. However, X11 display forwarding in SSHv2 (or other, secure protocols), combined with actual and pseudo-displays which accept connections only over local IPC mechanisms authorized by permissions or ACLs, does correct many X11 security problems as long as the "none" MAC is not used. It is RECOMMENDED that X11 display implementations default to allowing display opens only over local IPC. It is RECOMMENDED that SSHv2 server implementations that support X11 forwarding default to allowing display opens only over local IPC. On single-user systems it might be reasonable to default to allowing local display opens over TCP/IP.

Implementors of the X11 forwarding protocol SHOULD implement the magic cookie access checking spoofing mechanism as described in [ssh-connect] as an additional mechanism to prevent unauthorized use of the proxy.

Normative References

[SSH-ARCH]
Ylonen, T., "SSH Protocol Architecture", I-D draft-ietf-architecture-15.txt, Oct 2003.

[SSH-TRANS]
Ylonen, T., "SSH Transport Layer Protocol", I-D draft-ietf-transport-17.txt, Oct 2003.

[SSH-USERAUTH]
Ylonen, T., "SSH Authentication Protocol", I-D draft-ietf-userauth-18.txt, Oct 2003.

[SSH-CONNECT]

Ylonen, T., "SSH Connection Protocol", I-D draft-ietf-connect-18.txt, Oct 2003.

[SSH-NUMBERS] Lehtinen, S. and D. Moffat, "SSH Protocol Assigned Numbers", I-D draft-ietf-secsh-assignednumbers-05.txt, Oct 2003.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

Informative References

[FIPS-186] Federal Information Processing Standards Publication, "FIPS PUB 186, Digital Signature Standard", May 1994.

[FIPS-197] National Institue of Standards and Technology, "FIPS 197, Specification for the Advanced Encryption Standard", November 2001.

[ANSI T1.523-2001] American National Standards Insitute, Inc., "Telecom Glossary 2000", February 2001.

[SCHEIFLER] Scheifler, R., "X Window System : The Complete Reference to Xlib, X Protocol, Icccm, Xlfd, 3rd edition.", Digital Press ISBN 1555580882, Feburary 1992.

[RFC0854] Postel, J. and J. Reynolds, "Telnet Protocol Specification", STD 8, RFC 854, May 1983.

[RFC0894] Hornig, C., "Standard for the transmission of IP datagrams over Ethernet networks", STD 41, RFC 894, April 1984.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC1134] Perkins, D., "Point-to-Point Protocol: A proposal for multi-protocol transmission of datagrams over Point-to-Point links", RFC 1134, November 1989.

[RFC1282] Kantor, B., "BSD Rlogin", RFC 1282, December 1991.

[RFC1510] Kohl, J. and B. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510, September 1993.

[RFC1700] Reynolds, J. and J. Postel, "Assigned Numbers", RFC 1700, October 1994.

[RFC1750] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

[RFC3066] Alvestrand, H., "Tags for the Identification of Languages", BCP 47, RFC 3066, January 2001.

[RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996.

[RFC2025] Adams, C., "The Simple Public-Key GSS-API Mechanism (SPKM)", RFC 2025, October 1996.

[RFC2085] Oehler, M. and R. Glenn, "HMAC-MD5 IP Authentication with Replay Prevention", RFC 2085, February 1997.

[RFC2104] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2246] Dierks, T., Allen, C., Treese, W., Karlton, P., Freier, A. and P. Kocher, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2279] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[RFC2410] Glenn, R. and S. Kent, "The NULL Encryption Algorithm and Its Use With IPsec", RFC 2410, November 1998.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[SCHNEIER] Schneier, B., "Applied Cryptography Second Edition: protocols algorithms and source in code in C", 1996.

[KAUFMAN,PERLMAN,SPECINER] Kaufman, C., Perlman, R. and M. Speciner, "Network Security: PRIVATE Communication in a PUBLIC World", 1995.

[CERT] CERT Coordination Center, The., "http://www.cert.org/nav/

index_red.html".

[VENEMA] Venema, W., "Murphy's Law and Computer Security", Proceedings of 6th USENIX Security Symposium, San Jose CA http://www.usenix.org/publications/library/proceedings/sec96/venema.html, July 1996.

[ROGAWAY] Rogaway, P., "Problems with Proposed IP Cryptography", Unpublished paper http://www.cs.ucdavis.edu/˜rogaway/papers/draft-rogaway-ipsec-comments-00.txt, 1996.

[DAI] Dai, W., "An attack against SSH2 protocol", Email to the SECSH Working Group ietf-ssh@netbsd.org ftp://ftp.ietf.org/ietf-mail-archive/secsh/2002-02.mail, Feb 2002.

[BELLARE,KOHNO,NAMPREMPRE]
Bellaire, M., Kohno, T. and C. Namprempre, "Authenticated Encryption in SSH: Fixing the SSH Binary Packet Protocol", , Sept 2002.

Authors' Addresses

Tatu Ylonen
SSH Communications Security Corp
Fredrikinkatu 42
HELSINKI FIN-00100
Finland

EMail: ylo@ssh.com

Darren J. Moffat (editor)
Sun Microsystems, Inc
17 Network Circle
Menlo Park CA 94025
USA

EMail: Darren.Moffat@Sun.COM

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

The IETF has been notified of intellectual property rights claimed in regard to some or all of the specification contained in this document. For more information consult the online list of claimed rights.

Full Copyright Statement

Copyright (C) The Internet Society (2003). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assignees.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.