Network Working Group T. Ylonen
Internet-Draft SSH Communications Security Corp
Expires: March 2, 2003 D. Moffat, Ed.
Sun Microsystems, Inc
September 2002

SSH Authentication Protocol
draft-ietf-secsh-userauth-18.txt

Status of this Memo

This Internet-Draft will expire on March 2, 2003.

Copyright Notice

Abstract

SSH is a protocol for secure remote login and other secure network services over an insecure network. This document describes the SSH authentication protocol framework and public key, password, and host-based client authentication methods. Additional authentication methods are described in separate documents. The SSH authentication protocol runs on top of the SSH transport layer protocol and provides a single authenticated tunnel for the SSH connection protocol.

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

The SSH authentication protocol is a general-purpose user authentication protocol. It is intended to be run over the SSH transport layer protocol [SSH-TRANS]. This protocol assumes that the underlying protocols provide integrity and confidentiality protection.

This document should be read only after reading the SSH architecture document [SSH-ARCH]. This document freely uses terminology and notation from the architecture document without reference or further explanation.

The service name for this protocol is "ssh-userauth".

When this protocol starts, it receives the session identifier from the lower-level protocol (this is the exchange hash H from the first key exchange). The session identifier uniquely identifies this session and is suitable for signing in order to prove ownership of a private key. This protocol also needs to know whether the lower-level protocol provides confidentiality protection.

3. Conventions Used in This Document

The keywords "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", and "MAY" that appear in this document are to be interpreted as described in [RFC2119]

The used data types and terminology are specified in the architecture document [SSH-ARCH]

The architecture document also discusses the algorithm naming conventions that MUST be used with the SSH protocols.

3.1 The Authentication Protocol Framework

The server drives the authentication by telling the client which authentication methods can be used to continue the exchange at any given time. The client has the freedom to try the methods listed by the server in any order. This gives the server complete control over the authentication process if desired, but also gives enough flexibility for the client to use the methods it supports or that are most convenient for the user, when multiple methods are offered by the server.

Authentication methods are identified by their name, as defined in [SSH-ARCH]. The "none" method is reserved, and MUST NOT be listed as supported. However, it MAY be sent by the client. The server MUST always reject this request, unless the client is to be allowed in without any authentication, in which case the server MUST accept this request. The main purpose of sending this request is to get the list of supported methods from the server.

The server SHOULD have a timeout for authentication, and disconnect if the authentication has not been accepted within the timeout period. The RECOMMENDED timeout period is 10 minutes. Additionally, the implementation SHOULD limit the number of failed authentication attempts a client may perform in a single session (the RECOMMENDED limit is 20 attempts). If the threshold is exceeded, the server SHOULD disconnect.

3.1.1 Authentication Requests

All authentication requests MUST use the following message format. Only the first few fields are defined; the remaining fields depend on the authentication method.

```
  byte      SSH_MSG_USERAUTH_REQUEST
  string    user name (in ISO-10646 UTF-8 encoding [RFC2279])
  string    service name (in US-ASCII)
  string    method name (US-ASCII)
  The rest of the packet is method-specific.
```

The user name and service are repeated in every new authentication attempt, and MAY change. The server implementation MUST carefully check them in every message, and MUST flush any accumulated authentication states if they change. If it is unable to flush some authentication state, it MUST disconnect if the user or service name changes.

The service name specifies the service to start after authentication. There may be several different authenticated services provided. If the requested service is not available, the server MAY disconnect immediately or at any later time. Sending a proper disconnect message is RECOMMENDED. In any case, if the service does not exist, authentication MUST NOT be accepted.

If the requested user does not exist, the server MAY disconnect, or MAY send a bogus list of acceptable authentication methods, but never accept any. This makes it possible for the server to avoid disclosing information on which accounts exist. In any case, if the user does not exist, the authentication request MUST NOT be accepted.

While there is usually little point for clients to send requests that the server does not list as acceptable, sending such requests is not an error, and the server SHOULD simply reject requests that it does not recognize.

An authentication request MAY result in a further exchange of messages. All such messages depend on the authentication method used, and the client MAY at any time continue with a new SSH_MSG_USERAUTH_REQUEST message, in which case the server MUST abandon the previous authentication attempt and continue with the new one.

3.1.2 Responses to Authentication Requests

If the server rejects the authentication request, it MUST respond with the following:

```
byte      SSH_MSG_USERAUTH_FAILURE
string    authentications that can continue
boolean   partial success
```

"Authentications that can continue" is a comma-separated list of authentication method names that may productively continue the authentication dialog.

It is RECOMMENDED that servers only include those methods in the list that are actually useful. However, it is not illegal to include methods that cannot be used to authenticate the user.

Already successfully completed authentications SHOULD NOT be included in the list, unless they really should be performed again for some reason.

"Partial success" MUST be TRUE if the authentication request to which this is a response was successful. It MUST be FALSE if the request was not successfully processed.

When the server accepts authentication, it MUST respond with the following:

```
byte      SSH_MSG_USERAUTH_SUCCESS
```

Note that this is not sent after each step in a multi-method authentication sequence, but only when the authentication is complete.

The client MAY send several authentication requests without waiting for responses from previous requests. The server MUST process each request completely and acknowledge any failed requests with a SSH_MSG_USERAUTH_FAILURE message before processing the next request.

A request that results in further exchange of messages will be aborted by a second request. It is not possible to send a second request without waiting for a response from the server, if the first request will result in further exchange of messages. No SSH_MSG_USERAUTH_FAILURE message will be sent for the aborted method.

SSH_MSG_USERAUTH_SUCCESS MUST be sent only once. When SSH_MSG_USERAUTH_SUCCESS has been sent, any further authentication requests received after that SHOULD be silently ignored.

Any non-authentication messages sent by the client after the request that resulted in SSH_MSG_USERAUTH_SUCCESS being sent MUST be passed to the service being run on top of this protocol. Such messages can be identified by their message numbers (see Section Message Numbers (Section 3.2)).

3.1.3 The "none" Authentication Request

A client may request a list of authentication methods that may continue by using the "none" authentication method.

If no authentication at all is needed for the user, the server MUST return SSH_MSG_USERAUTH_SUCCESS. Otherwise, the server MUST return SSH_MSG_USERAUTH_FAILURE and MAY return with it a list of authentication methods that can continue.

This method MUST NOT be listed as supported by the server.

3.1.4 Completion of User Authentication

Authentication is complete when the server has responded with SSH_MSG_USERAUTH_SUCCESS; all authentication related messages received after sending this message SHOULD be silently ignored.

After sending SSH_MSG_USERAUTH_SUCCESS, the server starts the requested service.

3.1.5 Banner Message

In some jurisdictions, sending a warning message before authentication may be relevant for getting legal protection. Many UNIX machines, for example, normally display text from '/etc/issue', or use "tcp wrappers" or similar software to display a banner before issuing a login prompt.

The SSH server may send a SSH_MSG_USERAUTH_BANNER message at any time before authentication is successful. This message contains text to be displayed to the client user before authentication is attempted. The format is as follows:

```
byte      SSH_MSG_USERAUTH_BANNER
string    message (ISO-10646 UTF-8)
string    language tag (as defined in [RFC3066])
```

The client SHOULD by default display the message on the screen. However, since the message is likely to be sent for every login attempt, and since some client software will need to open a separate window for this warning, the client software may allow the user to explicitly disable the display of banners from the server. The message may consist of multiple lines.

If the message string is displayed, control character filtering discussed in [SSH-ARCH] SHOULD be used to avoid attacks by sending terminal control characters.

3.2 Authentication Protocol Message Numbers

All message numbers used by this authentication protocol are in the range from 50 to 79, which is part of the range reserved for protocols running on top of the SSH transport layer protocol.

Message numbers of 80 and higher are reserved for protocols running after this authentication protocol, so receiving one of them before authentication is complete is an error, to which the server MUST respond by disconnecting (preferably with a proper disconnect message sent first to ease troubleshooting).

After successful authentication, such messages are passed to the higher-level service.

These are the general authentication message codes:

```
#define SSH_MSG_USERAUTH_REQUEST            50
#define SSH_MSG_USERAUTH_FAILURE            51
#define SSH_MSG_USERAUTH_SUCCESS            52
```

```
#define SSH_MSG_USERAUTH_BANNER             53
```

In addition to the above, there is a range of message numbers (60..79) reserved for method-specific messages. These messages are only sent by the server (client sends only SSH_MSG_USERAUTH_REQUEST messages). Different authentication methods reuse the same message numbers.

3.3 Public Key Authentication Method: publickey

The only REQUIRED authentication method is public key authentication. All implementations MUST support this method; however, not all users need to have public keys, and most local policies are not likely to require public key authentication for all users in the near future.

With this method, the possession of a private key serves as authentication. This method works by sending a signature created with a private key of the user. The server MUST check that the key is a valid authenticator for the user, and MUST check that the signature is valid. If both hold, the authentication request MUST be accepted; otherwise it MUST be rejected. (Note that the server MAY require additional authentications after successful authentication.)

Private keys are often stored in an encrypted form at the client host, and the user must supply a passphrase before the signature can be generated. Even if they are not, the signing operation involves some expensive computation. To avoid unnecessary processing and user interaction, the following message is provided for querying whether authentication using the key would be acceptable.

```
byte      SSH_MSG_USERAUTH_REQUEST
string    user name
string    service
string    "publickey"
boolean   FALSE
string    public key algorithm name
string    public key blob
```

Public key algorithms are defined in the transport layer specification [SSH-TRANS]. The public key blob may contain certificates.

Any public key algorithm may be offered for use in authentication. In particular, the list is not constrained by what was negotiated during key exchange. If the server does not support some algorithm, it MUST simply reject the request.

The server MUST respond to this message with either

SSH_MSG_USERAUTH_FAILURE or with the following:

```
  byte      SSH_MSG_USERAUTH_PK_OK
  string    public key algorithm name from the request
  string    public key blob from the request
```

To perform actual authentication, the client MAY then send a signature generated using the private key. The client MAY send the signature directly without first verifying whether the key is acceptable. The signature is sent using the following packet:

```
  byte      SSH_MSG_USERAUTH_REQUEST
  string    user name
  string    service
  string    "publickey"
  boolean   TRUE
  string    public key algorithm name
  string    public key to be used for authentication
  string    signature
```

Signature is a signature by the corresponding private key over the following data, in the following order:

```
  string    session identifier
  byte      SSH_MSG_USERAUTH_REQUEST
  string    user name
  string    service
  string    "publickey"
  boolean   TRUE
  string    public key algorithm name
  string    public key to be used for authentication
```

When the server receives this message, it MUST check whether the supplied key is acceptable for authentication, and if so, it MUST check whether the signature is correct.

If both checks succeed, this method is successful. Note that the server may require additional authentications. The server MUST respond with SSH_MSG_USERAUTH_SUCCESS (if no more authentications are needed), or SSH_MSG_USERAUTH_FAILURE (if the request failed, or more authentications are needed).

The following method-specific message numbers are used by the publickey authentication method.

```
  /* Key-based */
  #define SSH_MSG_USERAUTH_PK_OK              60
```

3.4 Password Authentication Method: password

Password authentication uses the following packets. Note that a server MAY request the user to change the password. All implementations SHOULD support password authentication.

```
  byte      SSH_MSG_USERAUTH_REQUEST
  string    user name
  string    service
  string    "password"
  boolean   FALSE
  string    plaintext password (ISO-10646 UTF-8)
```

Note that the password is encoded in ISO-10646 UTF-8. It is up to the server how it interprets the password and validates it against the password database. However, if the client reads the password in some other encoding (e.g., ISO 8859-1 (ISO Latin1)), it MUST convert the password to ISO-10646 UTF-8 before transmitting, and the server MUST convert the password to the encoding used on that system for passwords.

Note that even though the cleartext password is transmitted in the packet, the entire packet is encrypted by the transport layer. Both the server and the client should check whether the underlying transport layer provides confidentiality (i.e., if encryption is being used). If no confidentiality is provided (none cipher), password authentication SHOULD be disabled. If there is no confidentiality or no MAC, password change SHOULD be disabled.

Normally, the server responds to this message with success or failure. However, if the password has expired the server SHOULD indicate this by responding with SSH_MSG_USERAUTH_PASSWD_CHANGEREQ. In anycase the server MUST NOT allow an expired password to be used for authentication.

```
  byte      SSH_MSG_USERAUTH_PASSWD_CHANGEREQ
  string    prompt (ISO-10646 UTF-8)
  string    language tag (as defined in [RFC3066])
```

In this case, the client MAY continue with a different authentication method, or request a new password from the user and retry password authentication using the following message. The client MAY also send this message instead of the normal password authentication request without the server asking for it.

```
  byte      SSH_MSG_USERAUTH_REQUEST
  string    user name
  string    service
```

```
string    "password"
boolean   TRUE
string    plaintext old password (ISO-10646 UTF-8)
string    plaintext new password (ISO-10646 UTF-8)
```

The server must reply to request message with SSH_MSG_USERAUTH_SUCCESS, SSH_MSG_USERAUTH_FAILURE, or another SSH_MSG_USERAUTH_PASSWD_CHANGEREQ. The meaning of these is as follows:

SSH_MSG_USERAUTH_SUCCESS The password has been changed, and authentication has been successfully completed.

SSH_MSG_USERAUTH_FAILURE with partial success The password has been changed, but more authentications are needed.

SSH_MSG_USERAUTH_FAILURE without partial success The password has not been changed. Either password changing was not supported, or the old password was bad. Note that if the server has already sent SSH_MSG_USERAUTH_PASSWD_CHANGEREQ, we know that it supports changing the password.

SSH_MSG_USERAUTH_CHANGEREQ The password was not changed because the new password was not acceptable (e.g. too easy to guess).

The following method-specific message numbers are used by the password authentication method.

```
#define SSH_MSG_USERAUTH_PASSWD_CHANGEREQ   60
```

3.5 Host-Based Authentication: hostbased

Some sites wish to allow authentication based on the host where the user is coming from, and the user name on the remote host. While this form of authentication is not suitable for high-security sites, it can be very convenient in many environments. This form of authentication is OPTIONAL. When used, special care SHOULD be taken to prevent a regular user from obtaining the private host key.

The client requests this form of authentication by sending the following message. It is similar to the UNIX "rhosts" and "hosts.equiv" styles of authentication, except that the identity of the client host is checked more rigorously.

This method works by having the client send a signature created with the private key of the client host, which the server checks with that host’s public key. Once the client host’s identity is established, authorization (but no further authentication) is performed based on the user names on the server and the client, and the client host name.

```
byte      SSH_MSG_USERAUTH_REQUEST
string    user name
string    service
string    "hostbased"
string    public key algorithm for host key
string    public host key and certificates for client host
string    client host name (FQDN; US-ASCII)
string    user name on the client host (ISO-10646 UTF-8)
string    signature
```

Public key algorithm names for use in "public key algorithm for host key" are defined in the transport layer specification. The "public host key for client host" may include certificates.

Signature is a signature with the private host key of the following data, in this order:

```
string    session identifier
byte      SSH_MSG_USERAUTH_REQUEST
string    user name
string    service
string    "hostbased"
string    public key algorithm for host key
string    public host key and certificates for client host
string    client host name (FQDN; US-ASCII)
string    user name on the client host(ISO-10646 UTF-8)
```

The server MUST verify that the host key actually belongs to the client host named in the message, that the given user on that host is allowed to log in, and that the signature is a valid signature on the appropriate value by the given host key. The server MAY ignore the client user name, if it wants to authenticate only the client host.

It is RECOMMENDED that whenever possible, the server perform additional checks to verify that the network address obtained from the (untrusted) network matches the given client host name. This makes exploiting compromised host keys more difficult. Note that this may require special handling for connections coming through a firewall.

4. Security Considerations

The purpose of this protocol is to perform client user authentication. It assumed that this runs 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. The transport layer provides forward secrecy for password authentication and other methods that rely on secret data.

Full security considerations for this protocol are provided in Section 8 of [SSH-ARCH]

Normative

[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

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

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

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 95025
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 (2002). 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.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgment

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