Network Working Group
INTERNET-DRAFT
draft-ietf-secsh-agent-02.txt
Expires: July 30, 2004

Tatu Ylonen
Timo J. Rinne
Sami Lehtinen
SSH Communications Security
30 January, 2004

Secure Shell Authentication Agent Protocol

Status of This Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Abstract

This document describes the Secure Shell authentication agent protocol (i.e., the protocol used between a client requesting authentication and the authentication agent). This protocol usually runs in a machine-specific local channel or over a forwarded authentication channel. It is assumed that the channel is trusted, so no protection for the communications channel is provided by this protocol.

Table of Contents

1. Authentication Agent Protocol

The authentication agent is a piece of software that runs in a user's local workstation, laptop, or other trusted device. It is used to implement single sign-on. It holds the user's private keys in its own storage, and can perform requested operations using the private key. It allows the keys to be kept on a smartcard or other special hardware that can perform cryptographic operations.

The authentication agent protocol is used to communicate between the authentication agent and clients wanting to authenticate something or wanting to perform private key operations.

The actual communication between the client and the agent happens using a machine-dependent trusted communications channel. This channel would typically be a local socket, named pipe, or some kind of secure

messaging system that works inside the local machine.

The protocol works by the client sending requests to the agent, and the agent responding to these requests.

1.1. Packet Format

All messages passed to/from the authentication agent have the following format:

```
  uint32           length
  byte             type
  data[length -1]  data payload
```

The following packet types are currently defined:

```
   /* Messages sent by the client. */
   #define SSH_AGENT_REQUEST_VERSION                       1
   #define SSH_AGENT_ADD_KEY                             202
   #define SSH_AGENT_DELETE_ALL_KEYS                     203
   #define SSH_AGENT_LIST_KEYS                           204
   #define SSH_AGENT_PRIVATE_KEY_OP                      205
   #define SSH_AGENT_FORWARDING_NOTICE                   206
   #define SSH_AGENT_DELETE_KEY                          207
   #define SSH_AGENT_LOCK                                208
   #define SSH_AGENT_UNLOCK                              209
   #define SSH_AGENT_PING                                212
   #define SSH_AGENT_RANDOM                              213

   #define SSH_AGENT_EXTENSION                           301

   /* Messages sent by the agent. */
   #define SSH_AGENT_SUCCESS                             101
   #define SSH_AGENT_FAILURE                             102
   #define SSH_AGENT_VERSION_RESPONSE                    103
   #define SSH_AGENT_KEY_LIST                            104
   #define SSH_AGENT_OPERATION_COMPLETE                  105
   #define SSH_AGENT_RANDOM_DATA                         106
   #define SSH_AGENT_ALIVE                               150
```

1.2. Forwarding Notices

If the agent connection is forwarded through intermediate hosts (using the SSH Connection Protocol agent forwarding feature (described in Section ``Agent Forwarding With Secure Shell'' of this document), or some other means), each intermediate node (Secure Shell client) should insert the following message into the agent channel before forwarding any other messages. The real agent will then receive these messages in sequence the nearest node first, and can determine whether the connection is from a local machine and if not, can log the path where the connection came from. These messages must be wrapped in the appropriate header.

```
  byte      SSH_AGENT_FORWARDING_NOTICE
  string    remote host name  (as typed by the user, preferably)
```

```
  string    remote host ip
  uint32    remote host port
```

1.3. Requesting Version Number

When the client opens a connection, it must send the following message to the server. This must be the first message sent. The real agent will receive this after zero or more forwarding notice messages.

```
byte      SSH_AGENT_REQUEST_VERSION
string    version string of the application sending the request
          (optional)
```

If the agent follows this protocol, it will respond with

```
byte      SSH_AGENT_VERSION_RESPONSE
uint32    version number, 3 for this protocol
<extension data>
```

If the version number request is ever sent to the Secure Shell 1.x agent, it will interpret it as a request to list identities. It will then respond with a message whose first byte is 2. This can be used to determine the version of the agent if compatibility with Secure Shell 1.x is desired.

If the version string query arrives without trailing string identifying the client software version, it can be translated list identities request sent by Secure Shell 1.x and handled accordingly. If agent software does not support the agent protocol of Secure Shell 1.x, it MAY also interpret this query as valid SSH_AGENT_REQUEST_VERSION packet.

The extension data in the SSH_AGENT_VERSION_RESPONSE may be empty, or may be a sequence of

```
string extension_name
string extension_data
```

pairs (both strings MUST always be present if one is, but the `extension_data' string may be of zero length). If present, these strings indicate extensions to the baseline protocol. The `extension_name' field(s) identify the name of the extension. The name should be of the form "name@domain", where the domain is the DNS domain name of the organization defining the extension. Additional names that are not of this format may be defined later by the IETF. Implementations MUST silently ignore any extensions whose name they do not recognize.

1.4. Adding Keys to the Agent

The client can add a new private key to the agent with the following message. Using this message over the net has security implications, and the implementation SHOULD warn the user before decryption or sending the private key. (XXX how does ssh-add detect this condition?)

```
  byte      SSH_AGENT_ADD_KEY
  string    private key encoding
  string    private key blob
```

```
  string    public key encoding
  string    public key and/or certificates for it
  string    description of the key
  ... 0, 1 or several constraints follow
```

1.4.1. Key types

Key blobs are preceeded by the encoding field, which defines how the blob should be interpreted. Defined values for public key encoding are "ssh-dss" and "ssh-rsa". Additional key types may be defined as specified in [SECSH-ARCH], under Section IANA Considerations (Section 8).

"ssh-dss" and "ssh-rsa" public key format encodings are defined in [SECSH-TRANS].

The "ssh-dss" private key format has the following specific encoding:

```
  string    "ssh-dss"
  mpint     p
  mpint     q
  mpint     g
  mpint     y
  mpint     x
```

The "ssh-rsa" private key format has the following specific encoding:

```
  string    "ssh-rsa"
  mpint     e
  mpint     d
  mpint     n
  mpint     u
  mpint     p
  mpint     q
```

XXX Additional key-types (for private keys), for example "ssh-rsa-encrypted"?

1.4.2. Forwarding constraints

All constraints are pairs of following format:

```
  byte      SSH_AGENT_CONSTRAINT_*
  variable  argument for the constraint
```

The type of the argument is dependent on the constraint type. Following constraint types are currently defined:

```
   /* Constraints 50-99 have a uint32 argument */
```

```
   /* Argument is uint32 defining key expiration time-out in
      seconds. After this timeout expires, the key can't be used.
      0 == no timeout */
```

```
   #define SSH_AGENT_CONSTRAINT_TIMEOUT                50

   /* Argument is uint32 defining the number of operations that can
      be performed with this key.  0xffffffff == no limit */
   #define SSH_AGENT_CONSTRAINT_USE_LIMIT              51

   /* Argument is uint32 defining the number of forwarding steps that
      this key can be forwarded.  0xffffffff == no limit */
   #define SSH_AGENT_CONSTRAINT_FORWARDING_STEPS       52

   /* Constraints 100-149 have a string argument */

   /* Argument is string defining the allowed forwarding steps for
      this key. XXX define this. */
   #define SSH_AGENT_CONSTRAINT_FORWARDING_PATH        100

   /* Constraints 150-199 have a boolean argument */

   /* Argument is a boolean telling whether the key can be used
      in Secure Shell 1.x compatibility operations. */

   #define SSH_AGENT_CONSTRAINT_SSH1_COMPAT            150

   /* Argument is a boolean telling whether operations performed
      with this key should  be confirmed interactively by the user
      or not. */
   #define SSH_AGENT_CONSTRAINT_NEED_USER_VERIFICATION 151
```

Message can contain zero, one or multiple constraints.

If the operation is successful, the agent will respond with the following message.

```
   byte      SSH_AGENT_SUCCESS
```

If the operation fails for some reason, the following message will be returned instead.

```
   byte      SSH_AGENT_FAILURE
   uint32    error code
   string    additional textual information (ISO-10646 UTF-8
             [RFC-2279])
   string    language tag (as defined in [RFC-1766])
```

The last two fields are optional; they don't need to be present in SSH_AGENT_FAILURE message. However, both MUST be provided if they are to be used. If client is version 2, the agent SHOULD NOT use these fields.

The error code is one of the following:

```
#define SSH_AGENT_ERROR_TIMEOUT              1
#define SSH_AGENT_ERROR_KEY_NOT_FOUND        2
#define SSH_AGENT_ERROR_DECRYPT_FAILED       3
```

```
   #define SSH_AGENT_ERROR_SIZE_ERROR            4
   #define SSH_AGENT_ERROR_KEY_NOT_SUITABLE      5
   #define SSH_AGENT_ERROR_DENIED                6
   #define SSH_AGENT_ERROR_FAILURE               7
   #define SSH_AGENT_ERROR_UNSUPPORTED_OP        8
```

1.5. Deleting Keys from the Agent

All keys that are in possession of the agent can be deleted with the following message. (The client is allowed to ignore this for some keys if desired.)

```
   byte      SSH_AGENT_DELETE_ALL_KEYS
```

The agent responds with either SSH_AGENT_SUCCESS or SSH_AGENT_FAILURE.

1.6. Deleting specific key from the Agent

The client can delete a specific key with given public key with following message.

```
   byte      SSH_AGENT_DELETE_KEY
   string    public key and/or certificates for it
   string    description of the key
```

The agent responds with either SSH_AGENT_SUCCESS or SSH_AGENT_FAILURE.

1.7. Listing the Keys that the Agent Can Use

The following message requests a list of all keys that the agent can use.

```
   byte      SSH_AGENT_LIST_KEYS
```

The agent will respond with the following message.

```
   byte      SSH_AGENT_KEY_LIST
   uint32    number_of_keys
   repeats number_of_keys times:
    string    public key blob or certificates
    string    description
```

2. Performing Private Key Operations

The real purpose of the agent is to perform private key operations. Such operations are performed with the following message.

```
   byte      SSH_AGENT_PRIVATE_KEY_OP
   string    operation name
   string    key or certificates, as returned in SSH_AGENT_KEY_LIST
```

```
   ... operation-specific data follows
```

The operation to be performed is identified by a name (string). Custom

operations can be added by suffixing the operation name by the fully qualified domain name of the person/organization adding the new operation.

When the operation is complete, the agent will respond with either SSH_AGENT_FAILURE or with the following message if the operation is successful:

```
byte      SSH_AGENT_OPERATION_COMPLETE
string    resulting data
```

If an operation is attempted that is not supported by the agent, the agent will respond with SSH_AGENT_FAILURE with error code set to SSH_AGENT_ERROR_UNSUPPORTED_OP.

The standard operations are defined below.

2.1. Signing

The agent can be used to create a digital signature using a key held by the agent. The operation name is "sign", and data in is a hash (suitable for the key) that is to be signed. This normally performs the raw private key operation, without hashing data first. The resulting data will be a binary representation of the output of the private key operation. The exact details of the operations to be performed depend on the key being used.

The operation-specific data has the following format:

```
string    data to be signed
```

Alternatively, it is possible to give the actual data to be signed to the agent. This is done using the operation "hash-and-sign". This is otherwise equal, but performs key-dependent hashing before signing.

If the requested operation is not legal for the key, SSH_AGENT_FAILURE will be returned with error code set to SSH_AGENT_ERROR_KEY_NOT_SUITABLE.

2.2. Decrypting

The agent can be used to decrypt a public key encrypted message with the operation "decrypt". This takes in raw public-key encrypted data, and returns the resulting decrypted data.

This may also fail. If the requested operation is not legal for the key, error code is set to SSH_AGENT_ERROR_KEY_NOT_SUITABLE.

The operation-specific data has the following format:

```
   string    data to be decrypted
```

2.3. Secure Shell Challenge-Response Authentication

Performs Secure Shell challenge-response authentication. This operation has the name "ssh1-challenge-response".

This operation works by first decrypting the challenge, then computing MD5 of the concatenation of the decrypted challenge and the session id (in this order), and returns the resulting 16 byte hash. The operation-specific data is in the following format:

```
string    challenge encrypted using the public key
string    session id
```

Normally, the length of the challenge before encryption will be 32 bytes and the length of the session id 16 bytes. The length of the encrypted challenge depends on the key and algorithm used.

3. Administrative Messages

There are also a number of messages that are only used to administer the agent. These might e.g. be used by a user interface for the agent. The agent should only allow these messages from local connection (i.e., if no forwarding notice messages were received before the version number request).

3.1. Locking and unlocking the agent

The agent can be temporarily locked by message:

```
byte      SSH_AGENT_LOCK
string    locking password
```

The agent responds with either SSH_AGENT_SUCCESS or SSH_AGENT_FAILURE. Particularily SSH_AGENT_FAILURE is sent, if agent is already locked. After this message, agent responds to all commands with SSH_AGENT_FAILURE until it receives a following command.

```
byte      SSH_AGENT_UNLOCK
string    locking password
```

The agent responds with either SSH_AGENT_SUCCESS or SSH_AGENT_FAILURE. Particularily SSH_AGENT_FAILURE is sent, if agent is not locked or if the submitted password does not match with one given with SSH_AGENT_LOCK message.

3.2. Miscellaneous Agent Commands

```
byte      SSH_AGENT_PING
... arbitrary padding data
```

Any agent or client receiving this message, should respond with

```
   byte      SSH_AGENT_ALIVE
```

```
   ... padding data from the SSH_AGENT_PING request
```

where the padding data is identical to the data sent with SSH_AGENT_PING.

```
   byte      SSH_AGENT_RANDOM
   uint32    the length of the requested random buffer
```

Client can request random data from the agent by this message. Agent responds either with SSH_AGENT_RANDOM_DATA or SSH_AGENT_FAILURE message.

```
   byte      SSH_AGENT_RANDOM_DATA
   string    random data
```

This message is a successful response to SSH_AGENT_RANDOM message. Message contains the random string of requested length.

4. Agent Forwarding With Secure Shell

The agent connection is typically forwarded over a Secure Shell connection. This requires small additions to the SSH Connection Protocol [SSH-CONN].

4.1. Requesting Agent Forwarding

Agent forwarding may be requested for a session by sending

```
  byte      SSH_MSG_CHANNEL_REQUEST
  uint32    recipient channel
  string    "auth-agent-req"
  boolean   want reply
```

This will, on success, create an agent listener to the remote end.

4.2. Agent Forwarding Channels

When a connection comes to the forwarded agent listener, a channel is opened to forward the connection to the other side.

```
  byte      SSH_MSG_CHANNEL_OPEN
  string    "auth-agent"
  uint32    sender channel
  uint32    initial window size
  uint32    maximum packet size
```

Implementations MUST reject these messages unless they have previously requested agent forwarding.

Forwarded agent channels are independent of any sessions, and closing a session channel does not in any way imply that forwarded connections

should be closed.

5. Vendor-Specific Extensions

The SSH_AGENT_EXTENSION request provides a generic extension mechanism for adding vendor-specific commands. The request has the following format:

```
  byte       SSH_AGENT_EXTENSION
  string     extension_id
  ... extension-specific data follows ...
```

`extension_id' is a string of the format "name@domain", where domain is an internet domain name of the vendor defining the request. The rest of the request is completely vendor-specific, and servers should only attempt to interpret it if they recognize the `extension_id' name.

These messages can be sent to either direction. However, the agent MUST send these messages only as responses to the client's requests. As an implementation note, the agent should use the standard responses if at all possible.

If the agent sees an extension message it doesn't understand, it should respond with SSH_AGENT_FAILURE with error SSH_AGENT_ERROR_UNSUPPORTED_OP.

6. Security Considerations

The authentication agent is used to control security-sensitive operations, and is used to implement single sign-on.

Anyone with access to the authentication agent can perform private key operations with the agent. This is a power equivalent to possession of the private key as long as the connection to the key is maintained. It is not possible to retrieve the key from the agent.

It is recommended that agent implementations allow and perform some form of logging and access control. This access control may utilize information about the path through which the connection was received (as collected with SSH_AGENT_FORWARDING_NOTICE messages; however, the path is reliable only up to and including the first unreliable machine.). Implementations should also allow restricting the operations that can be performed with keys - e.g., limiting them to challenge-response only.

One should note that a local superuser will be able to obtain access to agents running on the local machine. This cannot be prevented; in most operating systems, a user with sufficient privileges will be able to read the keys from the physical memory.

The authentication agent should not be run or forwarded to machine whose integrity is not trusted, as security on such machines might be compromised and might allow an attacker to obtain unauthorized access to the agent.

Adding a key with SSH_AGENT_ADD_KEY over the net (especially over the Internet) is generally not recommended, because at present the private

key has to be moved unencrypted. Implementations SHOULD warn the user of the implications. Even moving the key in encrypted form could be considered unwise.

7. Intellectual Property

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 implementers or users of this specification can be obtained from the IETF Secretariat.

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.

8. Additional Information

The current document editor is: Sami Lehtinen <sjl@ssh.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

9. Changes from previous versions

9.1. Changes between versions 3 and 2

o Added error message and language tag to SSH_AGENT_FAILURE.

o Added SSH_AGENT_EXTENSION.

o Added extension data to SSH_AGENT_VERSION_RESPONSE.

o Defined SSH_AGENT_ADD_KEY message better (previous version was underspecified).

10. References

Normative:

[SECSH-CONNECT] Ylonen, T., et al: "Secure Shell Connection Protocol", Internet-Draft, draft-ietf-secsh-connect-16.txt

[SECSH-TRANS] Ylonen, T., et al: "Secure Shell Transport Layer Protocol", Internet-Draft, draft-ietf-secsh-transport-10.txt

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

[RFC-1766] Alvestrand, H: "Tags for the Identification of Languages", March 1995.

Informative:

11. Address of Authors

Tatu Ylonen
SSH Communications Security Corp
Fredrikinkatu 42
FIN-00100 HELSINKI
Finland
E-mail: ylo@ssh.com

Timo J. Rinne
SSH Communications Security Corp
Fredrikinkatu 42
FIN-00100 HELSINKI
Finland
E-mail: tri@ssh.com

Sami Lehtinen
SSH Communications Security Corp
Fredrikinkatu 42
FIN-00100 HELSINKI
Finland
E-mail: sjl@ssh.com