Secure Shell Working Group
Internet-Draft
Expires: December 22, 2003

J. Galbraith
J. Van Dyke
B. McClure
VanDyke Software
June 23, 2003

Secure Shell Public-Key Subsystem
draft-galb-secsh-publickey-subsystem-02.txt

Status of this Memo

Copyright Notice

Abstract

SECSH defines an authentication mechanism that is based on public keys, but does not define any mechanism for key distribution. No common key management solution exists in current implementations. This document describes a protocol that can be used to configure public keys in an implementation-independent fashion, allowing client software to take on the burden of this configuration.

This protocol is intended to be used from the Secure Shell Connection Protocol [4] as a subsystem, as described in Section ``Starting a Shell or a Command''. The subsystem name used with this protocol is "publickey@vandyke.com".

The public-key subsystem provides a server-independent mechanism for clients to add public keys, remove public keys, and list the current public keys known by the server. Rights to manage public keys are specific and limited to the authenticated user.

A public key may also be associated with a mandatory command.

Table of Contents

1. Introduction

SECSH is a protocol for secure remote login and other secure network services over an insecure network. SECSH defines an authentication mechanism that is based on public keys, but does not define any mechanism for key distribution. Common practice is to authenticate once with password authentication and transfer the public key to the server. However, to date no two implementations use the same mechanism to configure a public key for use.

This document describes a subsystem that can be used to configure public keys in an implementation-independent fashion. This approach allows client software to take on the burden of this configuration. The public-key subsystem protocol is designed for extreme simplicity in implementation. It is not intended as a PKIX replacement.

The Secure Shell Public-Key subsystem has been designed to run on top of the SECSH transport layer [2] and user authentication protocols [3]. It provides a simple mechanism for the client to manage public keys on the server.

This document should be read only after reading the SECSH architecture [1] and SECSH connection [4] documents.

This protocol requires that the user be able to authenticate in some fashion before it can be used. If password authentication is used, servers SHOULD provide a configuration option to disable the use of password authentication after the first public key is added.

2. Public-Key Subsystem Overview

The public-key subsystem provides a server-independent mechanism for clients to add public keys, remove public keys, and list the current public keys known by the server. The subsystem name is "publickey@vandyke.com".

The public keys added, removed, and listed using this protocol are specific and limited to those of the authenticated user.

The operations to add, remove and list the authenticated user's public keys are performed as request packets sent to the server. The server sends response packets that indicate success or failure as well as provide specific response data.

The format of public-key blobs are detailed in the SSH Transport Protocol document [2].

2.1 Opening the Public-Key Subsystem

The public-key subsystem is opened when the clients sends a SSH_MSG_CHANNEL_REQUEST over an existing session.

The details of how a session is opened are described in the SSH Connection Protocol document [4] in the section "Opening a Session".

To open the public-key subsystem, the client sends:

```
byte      SSH_MSG_CHANNEL_REQUEST
uint32    recipient channel
string    "subsystem"
boolean   want reply
string    "publickey@vandyke.com"
```

Client implementations SHOULD reject this request; it is normally only sent by the client.

If want reply is TRUE, the server MUST respond with SSH_MSG_CHANNEL_SUCCESS if the public-key subsystem was successfully started or SSH_MSG_CHANNEL_FAILURE if the server failed to start or does not support the public-key subsystem.

The server SHOULD respond with SSH_MSG_CHANNEL_FAILURE if the user authenticated with a restricted public key that does not allow access to the publickey subsystem.

It is RECOMMENDED that clients request and check the reply for this request.

2.2 Requests

All public-key subsystem requests are sent in the following form:

```
uint32    length
string    request-name
... request specific data follows
```

The length field describes the length of the request-name field and the request-specific data, but not of the length field itself. The client MUST receive a response to each request prior to sending a new request.

All requests described in Section 3 are a description of the 'request-name' and 'data' portions of the packet.

2.3 Responses

All public-key subsystem responses are sent in the following form:

```
uint32    length
string    response-name
... response specific data follows
```

2.3.1 The Status Response

A request is acknowledged by sending a status packet. If there is data in response to the request, the status packet is sent after all data has been sent.

```
string    "status"
uint32    status code
string    description [RFC-2279]
string    language tag [RFC-1766]
```

A status message MUST be sent for any unrecognized packets and the request SHOULD NOT close the subsystem.

2.3.1.1 Status Codes

The status code gives the status in a more machine-readable format (suitable for localization), and can have the following values:

```
SSH_PUBLICKEY_SUCCESS                     0
SSH_PUBLICKEY_ACCESS_DENIED               1
SSH_PUBLICKEY_STORAGE_EXCEEDED            2
SSH_PUBLICKEY_REQUEST_NOT_SUPPORTED       3
SSH_PUBLICKEY_KEY_NOT_FOUND               4
```

```
        SSH_PUBLICKEY_KEY_NOT_SUPPORTED            5
      SSH_PUBLICKEY_KEY_ALREADY_PRESENT          6
```

SSH_PUBLICKEY_GENERAL_FAILURE 7

3. Public-Key Subsystem Operations

The public-key subsystem currently defines four operations: add, remove, list, and command.

3.1 Version Packet

Both sides MUST start by sending a version packet that indicates the version of the protocol they are using.

```
string "version"
uint32 protocol-version-number
```

The version of the protocol described by this document is version 1.

Both sides send the highest version that they implement. The lower of the version numbers is the version of the protocol to use. If either side can't support the lower version, it should close the subsystem and notify the other side by sending an SSH_MSG_CHANNEL_CLOSE message.

Both sides MUST wait to receive this version before continuing.

3.2 Adding a public key

If the client wishes to add a public key, the client sends:

```
string    "add"
string    public-key algorithm name
string    public-key blob
boolean   overwrite
uint32    attribute-count
          string attrib-name
          string attrib-value
          bool   mandatory
repeated attribute-count times
```

The server MUST attempt to store the public key for the user in the appropriate location so the public key can be used for subsequent public-key authentications. If the overwrite field is false and the specified key already exists, the server MUST return SSH_PUBLICKEY_KEY_ALREADY_PRESENT. If the server returns this, the client SHOULD provide an option to the user to overwrite the key. If the overwrite field is true and the specified key already exists but cannot be overwritten, the server MUST return SSH_PUBLICKEY_ACCESS_DENIED.

Attribute names are defined following the same scheme laid out for algorithm names in [SSH-ARCH] (section 5). If the server does not implement a mandatory attribute, it MUST fail the add. For the

purposes of a mandatory attribute, storage of the attribute is not sufficient, but requires that the server understand and implement the intent of the attribute.

The following attributes are currently defined:

"comment"
    The comment field contains user-specified text about the public key. The server SHOULD make every effort to preserve this value and return it with the key during a list operation. The server MUST NOT attempt to interpret or act upon the content of the comment field in any way.

    The comment field is useful so the user can identify the key without resorting to comparing its fingerprint.

    This attribute SHOULD NOT be mandatory.

"comment-language"
    If this attribute is specified, it MUST immediately follow a "comment" attribute and specifies the language for that attribute [RFC1766]. The client MAY specify more than comment if it additionally specifies a different language for each of those comments. The server SHOULD attempt to store each comment, together with that comment's lanuage attribute.

    This attribute SHOULD NOT be mandatory.

"command"
    "command" bypasses the session channel "exec" and "shell" requests by always executing the specified command (as if it had been executed using an "exec" request).

    This attribute SHOULD be mandatory. This attribute MUST NOT be specified if the "subsystem" attribute is specified.

"subsystem"
    "subsystem" specifies that the specified subsystem should be started when this key is used (as if it had been started using a "subsystem" request.

    This attribute SHOULD be mandatory. This attribute MUST NOT be specified if the "command" attribute is specified.

"restrict"
    The value of this attribute contains server functions that may not be performed when this key is used. It is a comma seperated list. Element names are specified in the same way as attribute names, above. The following restrictions are currently defined:

    Currently defined restrictions are:

```
"x11"
"shell"
"exec"
"agent"
"env"
"subsystem"
```

The "x11" restriction specifies that X11 forwarding may not be performed when this key is in use. The "shell" restriction specifies that session channel "shell" requests should be denied when this key is in use. The "exec" restriction specifies that session channel "exec" requests should be denied when this key is in use. The "agent" restriction specifies that session channel "auth-agent-req" requests should be denied when this key is in use. The "env" restriction specifies that session channel "env" requests should be denied when this key is in use. The "subsystem" restriction specifies that subsystems may not be started when this public key is in use (if the "subsystem" attribute is also specified, the subsystem specified in that attribute is exempted from this restriction).

This attribute SHOULD be mandatory.

"port-forward"
: "port-forward" specifies that no "direct-tcpip" requests should be accepted, except to those hosts specified in the comma-separated list supplied as a value to this attribute. If the value of this attribute is empty, all "direct-tcpip" requests should be refused when using this key.

  This attribute SHOULD be mandatory.

"reverse-forward"
: "reverse-forward" specifies that no "tcpip-forward" requests should be accepted, accept for the port numbers in the comma-separated list supplied as a value to this attribute. If the value of this attribute is empty, all "tcpip-forward" requests should be refused when using this key.

  This attribute SHOULD be mandatory.

In addition to the attributes and restrictions specified by the client, the server MAY provide a method for administrators to compulsorily enforce certain attributes or restrictions.

3.3 Removing a public key

If the client wishes to remove a public key, the client sends:

```
string    "remove"
string    public-key algorithm name
string    public-key blob
```

The server MUST attempt to remove the public key for the user from the appropriate location, so that the public key cannot be used for subsequent authentications.

3.4 Listing public keys

If the client wishes to list the known public keys, the client sends:

```
string    "list"
```

The server will respond with zero or more of the following responses:

```
string    "publickey"
string    public-key algorithm name
string    public-key blob
uint32    attribute-count
        string attrib-name
        string attrib-value
repeated attribute-count times
```

Following the last "publickey" response, a status packet MUST be sent.

An implementation MAY choose not to support this request.

3.5 Listing server capabilities

If the client wishes to know which restrictions the server supports, it sends:

```
string    "listattributes"
```

The server will respond with zero or more of the following responses:

```
string    "attribute"
string    attribute name
boolean   compulsory
```

The server will then respond with zero or more of the following responses:

```
string    "restriction"
string    restriction name
boolean   compulsory
```

The server MAY include "restrict" in the list of attributes it supports. The client SHOULD NOT require the server to do so in order to accept that the server supports the list of restrictions returned by the server.

Following the last "restriction" response, a status packet MUST be sent.

An implementation MAY choose not to support this request.

References

[1] Ylonen, T., Kivinen, T., Saarinen, M., Rinne, T. and S. Lehtinen, "SSH Protocol Architecture", draft-ietf-secsh-architecture-13 (work in progress), January 2002.

[2] Ylonen, T., Kivinen, T., Saarinen, M., Rinne, T. and S. Lehtinen, "SSH Transport Layer Protocol", draft-ietf-secsh-transport-15 (work in progress), March 2002.

[3] Ylonen, T., Kivinen, T., Saarinen, M., Rinne, T. and S. Lehtinen, "SSH Authentication Protocol", draft-ietf-secsh-userauth-16 (work in progress), February 2002.

[4] Ylonen, T., Kivinen, T., Saarinen, M., Rinne, T. and S. Lehtinen, "SSH Connection Protocol", draft-ietf-secsh-connect-16 (work in progress), January 2002.

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

Authors' Addresses

Joseph Galbraith
VanDyke Software
4848 Tramway Ridge Blvd
Suite 101
Albuquerque, NM 87111
US

Phone: +1 505 332 5700
EMail: galb-list@vandyke.com

Jeff P. Van Dyke
VanDyke Software
4848 Tramway Ridge Blvd
Suite 101
Albuquerque, NM 87111
US

Phone: +1 505 332 5700
EMail: jpv@vandyke.com

Brent McClure
VanDyke Software
4848 Tramway Ridge Blvd
Suite 101
Albuquerque, NM 87111
US

Phone: +1 505 332 5700
EMail: bdm@vandyke.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.

Full Copyright Statement

Acknowledgment

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