Network Working Group
Request for Comments: 4256
Category: Standards Track

F. Cusack
savecore.net
M. Forssen
AppGate Network Security AB

January 2006

Generic Message Exchange Authentication for the Secure Shell Protocol (SSH)

Status of This Memo

Copyright Notice

Abstract

The Secure Shell Protocol (SSH) is a protocol for secure remote login and other secure network services over an insecure network. This document describes a general purpose authentication method for the SSH protocol, suitable for interactive authentications where the authentication data should be entered via a keyboard (or equivalent alphanumeric input device). The major goal of this method is to allow the SSH client to support a whole class of authentication mechanism(s) without knowing the specifics of the actual authentication mechanism(s).

1. Introduction

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

This document describes a general purpose authentication method for the SSH authentication protocol. This method is suitable for interactive authentication methods that do not need any special software support on the client side. Instead, all authentication data should be entered via the keyboard. The major goal of this method is to allow the SSH client to have little or no knowledge of

the specifics of the underlying authentication mechanism(s) used by the SSH server. This will allow the server to arbitrarily select or change the underlying authentication mechanism(s) without having to update client code.

The name for this authentication method is "keyboard-interactive".

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

This document also describes some of the client interaction with the user in obtaining the authentication information. While this is somewhat out of the scope of a protocol specification, it is described here anyway because some aspects of the protocol are specifically designed based on user interface issues, and omitting this information may lead to incompatible or awkward implementations.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC-2119].

2. Rationale

Currently defined authentication methods for SSH are tightly coupled with the underlying authentication mechanism. This makes it difficult to add new mechanisms for authentication as all clients must be updated to support the new mechanism. With the generic method defined here, clients will not require code changes to support new authentication mechanisms, and if a separate authentication layer is used, such as [PAM], then the server may not need any code changes either.

This presents a significant advantage to other methods, such as the "password" method (defined in [SSH-USERAUTH]), as new (presumably stronger) methods may be added "at will" and system security can be transparently enhanced.

Challenge-response and One Time Password mechanisms are also easily supported with this authentication method.

However, this authentication method is limited to authentication mechanisms that do not require any special code, such as hardware drivers or password mangling, on the client.

3. Protocol Exchanges

The client initiates the authentication with an SSH_MSG_USERAUTH_REQUEST message. The server then requests authentication information from the client with an SSH_MSG_USERAUTH_INFO_REQUEST message. The client obtains the information from the user and then responds with an SSM_MSG_USERAUTH_INFO_RESPONSE message. The server MUST NOT send another SSH_MSG_USERAUTH_INFO_REQUEST before it has received the answer from the client.

3.1. Initial Exchange

The authentication starts with the client sending the following packet:

```
byte      SSH_MSG_USERAUTH_REQUEST
string    user name (ISO-10646 UTF-8, as defined in [RFC-3629])
string    service name (US-ASCII)
string    "keyboard-interactive" (US-ASCII)
string    language tag (as defined in [RFC-3066])
string    submethods (ISO-10646 UTF-8)
```

The language tag is deprecated and SHOULD be the empty string. It may be removed in a future revision of this specification. Instead, the server SHOULD select the language to be used based on the tags communicated during key exchange [SSH-TRANS].

If the language tag is not the empty string, the server SHOULD use the specified language for any messages sent to the client as part of this protocol. The language tag SHOULD NOT be used for language selection for messages outside of this protocol. If the server does not support the requested language, the language to be used is implementation-dependent.

The submethods field is included so the user can give a hint of which actual methods he wants to use. It is a comma-separated list of authentication submethods (software or hardware) that the user prefers. If the client has knowledge of the submethods preferred by the user, presumably through a configuration setting, it MAY use the submethods field to pass this information to the server. Otherwise, it MUST send the empty string.

The actual names of the submethods is something the user and the server need to agree upon.

Server interpretation of the submethods field is implementation-dependent.

One possible implementation strategy of the submethods field on the server is that, unless the user may use multiple different submethods, the server ignores this field. If the user may authenticate using one of several different submethods, the server should treat the submethods field as a hint on which submethod the user wants to use this time.

Note that when this message is sent to the server, the client has not yet prompted the user for a password, and so that information is NOT included with this initial message (unlike the "password" method).

The server MUST reply with an SSH_MSG_USERAUTH_SUCCESS, SSH_MSG_USERAUTH_FAILURE, or SSH_MSG_USERAUTH_INFO_REQUEST message.

The server SHOULD NOT reply with the SSH_MSG_USERAUTH_FAILURE message if the failure is based on the user name or service name; instead, it SHOULD send SSH_MSG_USERAUTH_INFO_REQUEST message(s), which look just like the one(s) that would have been sent in cases where authentication should proceed, and then send the failure message (after a suitable delay, as described below). The goal is to make it impossible to find valid usernames by comparing the results when authenticating as different users.

The server MAY reply with an SSH_MSG_USERAUTH_SUCCESS message if no authentication is required for the user in question. However, a better approach, for reasons discussed above, might be to reply with an SSH_MSG_USERAUTH_INFO_REQUEST message and ignore (don’t validate) the response.

3.2. Information Requests

Requests are generated from the server using the SSH_MSG_USERAUTH_INFO_REQUEST message.

The server may send as many requests as are necessary to authenticate the client; the client MUST be prepared to handle multiple exchanges. However, the server MUST NOT ever have more than one SSH_MSG_USERAUTH_INFO_REQUEST message outstanding. That is, it may not send another request before the client has answered.

The SSH_MSG_USERAUTH_INFO_REQUEST message is defined as follows:

```
byte      SSH_MSG_USERAUTH_INFO_REQUEST
string    name (ISO-10646 UTF-8)
string    instruction (ISO-10646 UTF-8)
string    language tag (as defined in [RFC-3066])
int       num-prompts
string    prompt[1] (ISO-10646 UTF-8)
boolean   echo[1]
...
string    prompt[num-prompts] (ISO-10646 UTF-8)
boolean   echo[num-prompts]
```

The language tag is deprecated and SHOULD be the empty string. It may be removed in a future revision of this specification. Instead, the server SHOULD select the language used based on the tags communicated during key exchange [SSH-TRANS].

If the language tag is not the empty string, the server SHOULD use the specified language for any messages sent to the client as part of this protocol. The language tag SHOULD NOT be used for language selection for messages outside of this protocol. If the server does not support the requested language, the language to be used is implementation-dependent.

The server SHOULD take into consideration that some clients may not be able to properly display a long name or prompt field (see next section), and limit the lengths of those fields if possible. For example, instead of an instruction field of "Enter Password" and a prompt field of "Password for user23@host.domain: ", a better choice might be an instruction field of "Password authentication for user23@host.domain" and a prompt field of "Password: ". It is expected that this authentication method would typically be backended by [PAM] and so such choices would not be possible.

The name and instruction fields MAY be empty strings; the client MUST be prepared to handle this correctly. The prompt field(s) MUST NOT be empty strings.

The num-prompts field may be ‘0’, in which case there will be no prompt/echo fields in the message, but the client SHOULD still display the name and instruction fields (as described below).

3.3. User Interface

Upon receiving a request message, the client SHOULD prompt the user as follows:

A command line interface (CLI) client SHOULD print the name and instruction (if non-empty), adding newlines. Then, for each prompt in turn, the client SHOULD display the prompt and read the user input.

A graphical user interface (GUI) client has many choices on how to prompt the user. One possibility is to use the name field (possibly prefixed with the application's name) as the title of a dialog window in which the prompt(s) are presented. In that dialog window, the instruction field would be a text message, and the prompts would be labels for text entry fields. All fields SHOULD be presented to the user. For example, an implementation SHOULD NOT discard the name field because its windows lack titles; instead, it SHOULD find another way to display this information. If prompts are presented in a dialog window, then the client SHOULD NOT present each prompt in a separate window.

All clients MUST properly handle an instruction field with embedded newlines. They SHOULD also be able to display at least 30 characters for the name and prompts. If the server presents names or prompts longer than 30 characters, the client MAY truncate these fields to the length it can display. If the client does truncate any fields, there MUST be an obvious indication that such truncation has occurred. The instruction field SHOULD NOT be truncated.

Clients SHOULD use control character filtering, as discussed in [SSH-ARCH], to avoid attacks by including terminal control characters in the fields to be displayed.

For each prompt, the corresponding echo field indicates whether the user input should be echoed as characters are typed. Clients SHOULD correctly echo/mask user input for each prompt independently of other prompts in the request message. If a client does not honor the echo field for whatever reason, then the client MUST err on the side of masking input. A GUI client might like to have a checkbox toggling echo/mask. Clients SHOULD NOT add any additional characters to the prompt, such as ": " (colon-space); the server is responsible for supplying all text to be displayed to the user. Clients MUST also accept empty responses from the user and pass them on as empty strings.

3.4. Information Responses

After obtaining the requested information from the user, the client MUST respond with an SSH_MSG_USERAUTH_INFO_RESPONSE message.

The format of the SSH_MSG_USERAUTH_INFO_RESPONSE message is as follows:

```
byte      SSH_MSG_USERAUTH_INFO_RESPONSE
int       num-responses
string    response[1] (ISO-10646 UTF-8)
...
string    response[num-responses] (ISO-10646 UTF-8)
```

Note that the responses are encoded in ISO-10646 UTF-8. It is up to the server how it interprets the responses and validates them. However, if the client reads the responses in some other encoding (e.g., ISO 8859-1), it MUST convert the responses to ISO-10646 UTF-8 before transmitting.

From an internationalization standpoint, it is desired that if a user enters responses, the authentication process will work regardless of what OS and client software they are using. Doing so requires normalization. Systems supporting non-ASCII passwords SHOULD always normalize passwords and usernames whenever they are added to the database, or compare them (with or without hashing) to existing entries in the database. SSH implementations that both store the passwords and compare them SHOULD use [SASLPREP] for normalization.

If the num-responses field does not match the num-prompts field in the request message, the server MUST send a failure message.

In the case that the server sends a ‘0’ num-prompts field in the request message, the client MUST send a response message with a ‘0’ num-responses field to complete the exchange.

The responses MUST be ordered as the prompts were ordered. That is, response[n] MUST be the answer to prompt[n].

After receiving the response, the server MUST send either an SSH_MSG_USERAUTH_SUCCESS, SSH_MSG_USERAUTH_FAILURE, or another SSH_MSG_USERAUTH_INFO_REQUEST message.

If the server fails to authenticate the user (through the underlying authentication mechanism(s)), it SHOULD NOT send another request message(s) in an attempt to obtain new authentication data; instead, it SHOULD send a failure message. The only time the server should send multiple request messages is if additional authentication data

is needed (i.e., because there are multiple underlying authentication mechanisms that must be used to authenticate the user).

If the server intends to respond with a failure message, it MAY delay for an implementation-dependent time before sending it to the client. It is suspected that implementations are likely to make the time delay configurable; a suggested default is 2 seconds.

4. Authentication Examples

Here are two example exchanges between a client and server. The first is an example of challenge/response with a handheld token. This is an authentication that is not otherwise possible with other authentication methods.

```
C:   byte      SSH_MSG_USERAUTH_REQUEST
C:   string    "user23"
C:   string    "ssh-userauth"
C:   string    "keyboard-interactive"
C:   string    ""
C:   string    ""

S:   byte      SSH_MSG_USERAUTH_INFO_REQUEST
S:   string    "CRYPTOCard Authentication"
S:   string    "The challenge is '14315716'"
S:   string    "en-US"
S:   int       1
S:   string    "Response: "
S:   boolean   TRUE

[Client prompts user for password]

C:   byte      SSH_MSG_USERAUTH_INFO_RESPONSE
C:   int       1
C:   string    "6d757575"

S:   byte      SSH_MSG_USERAUTH_SUCCESS
```

The second example is a standard password authentication; in this case, the user's password is expired.

```
C:   byte      SSH_MSG_USERAUTH_REQUEST
C:   string    "user23"
C:   string    "ssh-userauth"
C:   string    "keyboard-interactive"
C:   string    "en-US"
C:   string    ""

S:   byte      SSH_MSG_USERAUTH_INFO_REQUEST
S:   string    "Password Authentication"
S:   string    ""
S:   string    "en-US"
S:   int       1
S:   string    "Password: "
S:   boolean   FALSE

[Client prompts user for password]

C:   byte      SSH_MSG_USERAUTH_INFO_RESPONSE
C:   int       1
C:   string    "password"

S:   byte      SSH_MSG_USERAUTH_INFO_REQUEST
S:   string    "Password Expired"
S:   string    "Your password has expired."
S:   string    "en-US"
S:   int       2
S:   string    "Enter new password: "
S:   boolean   FALSE
S:   string    "Enter it again: "
S:   boolean   FALSE

[Client prompts user for new password]

C:   byte      SSH_MSG_USERAUTH_INFO_RESPONSE
C:   int       2
C:   string    "newpass"
C:   string    "newpass"

S:   byte      SSH_MSG_USERAUTH_INFO_REQUEST
S:   string    "Password changed"
S:   string    "Password successfully changed for user23."
S:   string    "en-US"
S:   int       0
```

```
      [Client displays message to user]

      C:   byte      SSH_MSG_USERAUTH_INFO_RESPONSE
      C:   int       0

      S:   byte      SSH_MSG_USERAUTH_SUCCESS
```

5. IANA Considerations

The userauth type "keyboard-interactive" is used for this authentication method.

The following method-specific constants are used with this authentication method:

```
   SSH_MSG_USERAUTH_INFO_REQUEST           60
   SSH_MSG_USERAUTH_INFO_RESPONSE          61
```

6. Security Considerations

The authentication protocol and this authentication method depend on the security of the underlying SSH transport layer. Without the confidentiality provided therein, any authentication data passed with this method is subject to interception.

The number of client-server exchanges required to complete an authentication using this method may be variable. It is possible that an observer may gain valuable information simply by counting that number. For example, an observer may guess that a user's password has expired, and with further observation may be able to determine the password lifetime imposed by a site's password expiration policy.

7. References

7.1. Normative References

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

[RFC-3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

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

[SSH-ARCH] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

[SSH-USERAUTH] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, January 2006.

[SSH-TRANS] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, January 2006.

[SASLPREP] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.

7.2. Informative References

[PAM] Samar, V., Schemers, R., "Unified Login With Pluggable Authentication Modules (PAM)", OSF RFC 86.0, October 1995.

Authors' Addresses

Frank Cusack
savecore.net

EMail: frank@savecore.net

Martin Forssen
AppGate Network Security AB
Otterhallegatan 2
SE-411 18 Gothenburg
SWEDEN

EMail: maf@appgate.com

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).