Internet-Draft
Updates: 4252, 4253, 4254 (if approved)
Intended status: Standards Track
Expires: March 5, 2017

D. Bider
Bitvise Limited

September 5, 2016

Extension Negotiation in Secure Shell (SSH)
draft-ietf-curdle-ssh-ext-info-01.txt

Abstract

This memo defines a mechanism for SSH clients and servers to exchange information about supported protocol extensions confidentially after completed key exchange.

Status

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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/1id-abstracts.html

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

1. Overview and Rationale

Secure Shell (SSH) is a common protocol for secure communication on the Internet. The original design of the SSH transport layer [RFC4253] lacks proper extension negotiation. Meanwhile, diverse implementations take steps to ensure that known message types contain no unrecognized information. This makes it difficult for implementations to signal capabilities and negotiate extensions without risking disconnection.

This obstacle has been recognized in relationship with [SSH-RSA-SHA2], where the need arises for a client to discover signature algorithms a server accepts, to avoid authentication penalties and trial-and-error.

1.1. Requirements Terminology

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 [RFC2119].

2. Extension Negotiation Mechanism

2.1. Signaling of Extension Negotiation in KEXINIT

Applications implementing this mechanism MUST add to the field "kex_algorithms", in their KEXINIT packet sent for the first key exchange, one of the following indicator names:

- When acting as server: "ext-info-s"
- When acting as client: "ext-info-c"

The indicator name is added without quotes, and MAY be added at any position in the name-list, subject to proper separation from other names as per name-list conventions.

The names are added to the "kex_algorithms" field because this is one of two name-list fields in KEXINIT that do not have a separate copy for each data direction.

The indicator names inserted by the client and server are different to

ensure that these names will not produce a match, and will be neutral with respect to key exchange algorithm negotiation.

The inclusion of textual indicator names is intended to provide a clue for implementers to discover this mechanism.

2.2. Enabling Criteria

If a client or server offers "ext-info-c" or "ext-info-s" respectively, it must be prepared to accept an SSH_MSG_EXT_INFO message from the peer.

Thus a server only needs to send "ext-info-s" if it intends to process SSH_MSG_EXT_INFO from the client.

If a server receives an "ext-info-c", it MAY send an SSH_MSG_EXT_INFO message, but is not required to do so.

If an SSH_MSG_EXT_INFO message is sent, then it MUST be the first message after the initial SSH_MSG_NEWKEYS.

Implementations MUST NOT send an incorrect indicator name for their role. Implementations MAY disconnect if the counter-party sends an incorrect indicator. If "ext-info-c" or "ext-info-s" ends up being negotiated as a key exchange method, the parties MUST disconnect.

2.3. SSH_MSG_EXT_INFO Message

A party that received the "ext-info-c" or "ext-info-s" indicator MAY send the following message:

```
byte       SSH_MSG_EXT_INFO (value 7)
uint32     nr-extensions
repeat "nr-extensions" times:
  string   extension-name
  string   extension-value
```

This message is sent without delay, and immediately after SSH_MSG_NEWKEYS.

2.4. Server's Secondary SSH_MSG_EXT_INFO

If the client sent "ext-info-c", the server MAY send, but is not

obligated to send, an SSH_MSG_EXT_INFO message immediately before SSH_MSG_USERAUTH_SUCCESS, as defined in [RFC4252]. The server MAY send this message whether or not it sent EXT_INFO after SSH_MSG_NEWKEYS.

This allows a server to reveal support for additional extensions that it was unwilling to reveal to an unauthenticated client. If a server sends a subsequent SSH_MSG_EXT_INFO, this replaces any initial one, and both the client and the server re-evaluate extensions in effect. The server's last EXT_INFO is matched against the client's original.

2.5. Interpretation of Extension Names and Values

Each extension is identified by its extension-name, and defines the conditions under which the extension is considered to be in effect. Applications MUST ignore unrecognized extension-names.

In general, if an extension requires both the client and the server to include it in order for the extension to take effect, the relative position of the extension-name in each EXT_INFO message is irrelevant.

Extension-value fields are interpreted as defined by their respective extension. An extension-value field MAY be empty if so permitted by the extension. Applications that do not implement or recognize a particular extension MUST ignore the associated extension-value field, regardless of its size or content.

The cumulative size of an SSH_MSG_EXT_INFO message is limited only by the maximum packet length that an implementation may apply in accordance with [RFC4253]. Implementations MUST accept well-formed SSH_MSG_EXT_INFO messages up to the maximum packet length they accept.

3. Initially Defined Extensions

3.1. "server-sig-algs"

This extension is sent with the following extension name and value:

```
string      "server-sig-algs"
name-list   signature-algorithms-accepted
```

Note that the name-list type is a strict subset of the string type,

and is thus permissible as an extension-value.

This extension is sent by the server only, and contains a list of signature algorithms that the server is able to process as part of a "publickey" request.

A client that wishes to proceed with public key authentication MAY wait for the server's SSH_MSG_EXT_INFO so it can send a "publickey" authentication request with an appropriate signature algorithm, rather than resorting to trial and error.

Servers that implement public key authentication SHOULD implement this extension.

If a server does not send this extension, a client SHALL NOT make any assumptions about the server's signature algorithm support, and MAY proceed with authentication request trial and error.

3.2. "no-flow-control"

This extension is sent with the following extension name and value:

```
string      "no-flow-control"
string      (empty)
```

This extension MUST be sent by both parties in order to take effect.

If included by both parties, the effect of this extension is that the "initial window size" fields in the messages SSH_MSG_CHANNEL_OPEN and

SSH_MSG_CHANNEL_OPEN_CONFIRMATION, as defined in [RFC4254], become meaningless. The values of these fields MUST be ignored, and a channel behaves as if the window size in either direction is infinite. Neither side is required to send any SSH_MSG_CHANNEL_WINDOW_ADJUST messages, and if received, such messages MUST be ignored.

This extension is intended, but not limited to, use by file transfer applications that are only going to use one channel, and for which the flow control provided by SSH is an impediment, rather than a feature.

Implementations MUST refuse to open more than one simultaneous channel when this extension is in effect. Nevertheless, server implementations SHOULD support clients opening more than one non-simultaneous channel.

3.3. "accept-channels"

This extension is sent with the following extension name and value:

```
string      "accept-channels"
name-list   channel-types-accepted
```

An implementation MAY use this extension to signal to the other party a list of channel types it might accept. A server that adapts the list of available channel types based on authentication MAY defer sending this extension until a subsequent EXT_INFO, just before sending the message USERAUTH_SUCCESS.

An implementation is not obligated to unconditionally accept open requests for channel types advertised in this extension. An open request for a listed channel type MAY still fail for another reason.

3.4. "elevation"

This extension MAY be sent by the client as follows:

```
string      "elevation"
string      choice of: "y" | "n" | "d"
```

A client sends "y" to indicate its preference that the session should be elevated (as used by Windows); "n" to not be elevated; and "d" for the server to use its default behavior. If a client does not send the "elevation" extension, the server SHOULD act as if "d" was sent.

If a client has included this extension, then after authentication, a server that supports this extension SHOULD indicate to the client whether elevation was done by sending the following global request:

```
byte        SSH_MSG_GLOBAL_REQUEST
string      "elevation"
boolean     want reply = false
boolean     elevation performed
```


3.5. "delay-compression"

This extension MAY be sent by both parties as follows:

```
  string          "delay-compression"
  string:
    name-list     compression_algorithms_client_to_server
    name-list     compression_algorithms_server_to_client
```

This extension allows the server and client to renegotiate compression algorithm support without having to conduct a key re-exchange, putting new algorithms into effect immediately upon successful authentication.

This extension takes effect only if both parties send it. Name-lists MAY include any compression algorithm that could have been negotiated in SSH_MSG_KEXINIT, except algorithms that define their own delayed compression semantics. This means "zlib,none" is a valid algorithm list in this context; but "zlib@openssh.com" is not.

If both parties send this extension, but the name-lists do not contain a common algorithm in either direction, the parties MUST disconnect in the same way as if negotiation failed as part of SSH_MSG_KEXINIT.

If this extension takes effect, the renegotiated compression algorithm is used as follows:

- By the server, starting with the very next SSH message after SSH_MSG_USERAUTH_SUCCESS.

- By the client, after sending SSH_MSG_NEWCOMPRESS. If this extension takes effect, the client MUST send the following message immediately after receiving the server's SSH_MSG_USERAUTH_SUCCESS:

  ```
    byte       SSH_MSG_NEWCOMPRESS (value 8)
  ```

  The purpose of this message is to avoid a race condition where the server cannot reliably know whether a message sent by the client was sent before or after receiving the server's USERAUTH_SUCCESS.

As with all extensions, the server may delay including this extension until its secondary SSH_MSG_EXT_INFO, sent before USERAUTH_SUCCESS. This allows the server to avoid advertising compression support until the client has been authenticated.

In subsequent key re-exchange, the compression algorithms negotiated in re-exchange override the algorithms negotiated with this extension.

4. IANA Considerations

4.1. Additions to existing tables

IANA is requested to insert the following entries into the table Message Numbers under Secure Shell (SSH) Protocol Parameters [RFC4250]:

Value	Message ID	Reference
7	SSH_MSG_EXT_INFO	[this document]
8	SSH_MSG_NEWCOMPRESS	[this document]

IANA is requested to insert the following entries into the table Key Exchange Method Names:

Method Name	Reference	Note
ext-info-s	[this document]	Section 2.2
ext-info-c	[this document]	Section 2.2

4.2. New table: Extension Names

Also under Secure Shell (SSH) Protocol Parameters, IANA is requested to create a new table, Extension Names, with initial content:

Extension Name	Reference	Note
server-sig-algs	[this document]	Section 3.1
no-flow-control	[this document]	Section 3.2
accept-channels	[this document]	Section 3.3
elevation	[this document]	Section 3.4
delay-compression	[this document]	Section 3.5

4.2.1. Future Assignments to Extension Names

Names in the Extension Names table MUST follow the Conventions for Names defined in [RFC4250], Section 4.6.1.

Requests for assignments of new non-local names in the Extension Names table (i.e. names not including the '@' character) MUST be done through the IETF CONSENSUS method, as described in [RFC5226].

6. References

6.1. Normative References

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

[RFC4250] Lehtinen, S. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Assigned Numbers", RFC 4250, January 2006.

[RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, January 2006.

[RFC4253] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Transport Layer Protocol", RFC 4253, January 2006.

[RFC4254] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Connection Protocol", RFC 4254, January 2006.

[RFC5226] Narten, T. and Alvestrand, H., "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

6.2. Informative References

[SSH-RSA-SHA2] Bider, D., "Use of RSA Keys with SHA-2 256 and 512 in Secure Shell (SSH)", draft-ietf-curdle-rsa-sha2-01.txt, August 2016, <https://tools.ietf.org/html/draft-ietf-curdle-rsa-sha2-01>.

Author's Address

Denis Bider
Bitvise Limited
Suites 41/42, Victoria House
26 Main Street
GI

Phone: +506 8315 6519
EMail: ietf-ssh3@denisbider.com
URI: https://www.bitvise.com/

Acknowledgments

Thanks to Markus Friedl and Damien Miller for comments and initial implementation.