Extension Negotiation in Secure Shell (SSH)
draft-ssh-ext-info-00.txt

Abstract
This memo defines a mechanism for SSH clients and servers to exchange information about supported protocol extensions confidentially after completed key exchange. This is done while optimizing protocol startup to eliminate the round-trip the original protocol requires on messages SSH_MSG_SERVICE_REQUEST and SSH_MSG_SERVICE_ACCEPT.

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

Copyright
Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
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 efficiently discover signature algorithms a server accepts, to avoid round-trips of trial and error.

The design of SSH as-is also spends an unnecessary roundtrip on the client sending an SSH_MSG_SERVICE_REQUEST and awaiting an SSH_MSG_SERVICE_ACCEPT in reply. This was designed envisioning a possible SSH service other than "ssh-userauth"; but a decade later, there is no such service. (If it were to arise, negotiation for such a service can also be supported using the mechanism in this document.)

This memo describes an extension negotiation mechanism that replaces SSH_MSG_SERVICE_REQUEST and ACCEPT, and meets the above needs.

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. Discussion of Extension Options

The SSH protocol contains multiple potential avenues for extension:

a) By adding fields to existing packets.

If existing implementations permitted, it would have been possible to add additional fields to the currently not-most-useful SERVICE_REQUEST and SERVICE_ACCEPT messages. Such additional fields could serve for extension negotiation.

However, a number of implementations, including OpenSSH, have chosen a restrictive view of packet formats, and will disconnect if additional fields are included in known packet types.

b) By designating a use for the "reserved" field in SSH_MSG_KEXINIT.

[RFC4253] defines the following field at the end of KEXINIT:

\[
\text{uint32} \quad 0 \text{ (reserved for future extension)}
\]

A logical interpretation of this definition would be that applications must send this field as zero, but must accept values other than zero. Otherwise, this field could not be useful for "future extension".
Unfortunately, current non-use of this field makes it likely that there will be implementations that do not handle unexpected values correctly. This is the case with current latest versions of libssh, which do not decode this field when it is received, but instead assume it is zero without reading it. Key exchange then fails if the field is non-zero, because the parties are hashing different KEXINIT values.

c) By incrementing the protocol version number.

[RFC4253] includes the following statement:

Since the protocol being defined in this set of documents is version 2.0, the ‘protoversion’ MUST be "2.0".

The RFC then goes on to discuss 1.xx compatibility cases when the version does not have to be "2.0", after all.

The author has investigated the source code of several implementations (OpenSSH 7.1; PuTTY 0.64; libssh 0.7.2; the author’s own Bitvise SSH) and has not found evidence that any of these implementations would reject a counterparty that advertises a protocol version number incremented from "2.0" to e.g. "2.1".

However, the author cannot prove there are not other, more restrictive implementations. Applications that could have trouble with a changed protocol version may include not only SSH clients and servers, but also applications that interact with SSH, including security scanners; load balancers; or deep packet inspection firewalls.

The author perceives that the protocol version number is too central and fundamental to be incremented for a change that does not affect security, and does not involve a change in KEXINIT itself.

d) By defining special names to be used with algorithm negotiation.

This is the only extension method known to the author which appears to have guaranteed compatibility with existing applications; without side effects; and has the added bonus of providing clues to implementers about what to search for, to learn about the extension.

The author has chosen this method for the mechanism described herein.
3. Extension Negotiation Mechanism

3.1. 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 suggested position is last in the list, but implementations MUST recognize these indicator names when included at any position.

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; "server_host_key_algorithms" being the other. Of the two fields, "server_host_key_algorithms" has special cases for negotiation that the "kex_algorithms" field does not. This makes "kex_algorithms" a more dependable candidate for this mechanism.

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.

3.2. Mechanism Enabling Criteria

The mechanism described in this memo is enabled when all of the following conditions are true:

- The server and client have completed one or more key exchanges.

- The KEXINIT packet sent by the server during the LAST key exchange included the indicator name "ext-info-s" at any position in the "kex_algorithms" field (subject to name-list conventions).

- The KEXINIT packet sent by the client during the SAME key exchange included the indicator name "ext-info-c" at any position in the "kex_algorithms" field (subject to name-list conventions).

Once the mechanism has been enabled, it remains enabled for the rest of the SSH session. Once enabled, both parties MUST continue to send their respective indicator name in subsequent key exchanges. However, both parties MUST ignore the presence or absence of these indicator names in subsequent key exchanges once the mechanism has been enabled.
A server MUST include its indicator name in the first key exchange.

A client MAY omit its indicator name in the first key exchange. After this key exchange, the client MAY immediately initiate a subsequent key exchange, in which it includes its indicator. If the server also includes its indicator in this subsequent exchange, the mechanism is enabled starting from completion of that key exchange.

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

3.3. Actions after Mechanism is Enabled

If a party has already sent SERVICE_REQUEST or SERVICE_ACCEPT at the time this mechanism has become enabled, the state of the mechanism is undefined, and the party MUST disconnect.

Otherwise, immediately after sending SSH_MSG_NEWKEYS for the key exchange in which the mechanism was first enabled, each party sends the following message, replacing SERVICE_REQUEST and SERVICE_ACCEPT:

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

Each party sends this message without delay, and immediately after SSH_MSG_NEWKEYS; but allowing for any intervening SSH_MSG_IGNORE or DEBUG, which MUST be either handled appropriately or discarded. This EXT_INFO message is sent by the client at most once in an SSH session, and by the server one or more times before SSH_MSG_USERAUTH_SUCCESS.

The SSH_MSG_EXT_INFO message replaces both the SERVICE_REQUEST and SERVICE_ACCEPT messages that might have otherwise been sent by either party. Enabling this mechanism implies the same effects as if the client had sent a SERVICE_REQUEST for the service "ssh-userauth", and as if the server had responded with SERVICE_ACCEPT.

Since SSH_MSG_EXT_INFO is sent immediately by both parties, and replaces SSH_MSG_SERVICE_REQUEST and SSH_MSG_SERVICE_ACCEPT, the client MAY follow immediately with SSH_MSG_USERAUTH_REQUEST; or it MAY wait for the server’s SSH_MSG_EXT_INFO before starting authentication.

3.4. Server’s Secondary SSH_MSG_EXT_INFO

The client sends its SSH_MSG_EXT_INFO only once per session, immediately after the key exchange that enabled this mechanism.
The server MUST send its first SSH_MSG_EXT_INFO at the same logical time as the client, immediately after the key exchange that first enabled this mechanism.

In addition, the server MAY send, but is not obligated to send, one or more additional SSH_MSG_EXT_INFO messages at any time before sending the message SSH_MSG_USERAUTH_SUCCESS, as defined in [RFC4252].

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 previous ones, 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.

A server MUST NOT send another SSH_MSG_EXT_INFO message after sending SSH_MSG_USERAUTH_SUCCESS.

3.5. 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.
4. Initially Defined Extensions

4.1. "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" authentication 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.

If a server sends this extension with an empty algorithm list, the client SHOULD take this to mean that the server will not accept any public key authentication request; even if the "publickey" method is included by the server in the "authentications that can continue" field in the message SSH_MSG_USERAUTH_FAILURE.

Note that the server’s list of accepted signature algorithms MAY change in subsequent EXT_INFO messages sent before the end of user authentication.

4.2. "client-req-ok"

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

    string    "client-req-ok"
    string    (empty)

This extension is sent by the client only, and indicates that the client implementation will gracefully handle a global request; especially an unrecognized global request.
All clients SHOULD send this extension, and SHOULD gracefully handle unrecognized global requests. This allows the server to use an active keep-alive strategy whereby a global request is periodically sent to the client, to which the client is expected to reply to indicate the connection is active.

In the absence of this extension, a server cannot assume that the client won’t disconnect when receiving a global request, since there are existing deployed clients that do so.

4.3. "no-handbrake"

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

```plaintext
string      "no-handbrake"
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 MAY refuse to open more than one simultaneous channel when this extension is in effect. However, server implementations SHOULD support clients opening more than one consecutive channel.
5. IANA Considerations

5.1. Additions to existing tables

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Message ID</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>SSH_MSG_EXT_INFO</td>
<td>[this document]</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Reference</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext-info-s</td>
<td>[this document]</td>
<td>Section 2.2</td>
</tr>
<tr>
<td>ext-info-c</td>
<td>[this document]</td>
<td>Section 2.2</td>
</tr>
</tbody>
</table>

5.2. New table: Extension Names

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

<table>
<thead>
<tr>
<th>Extension Name</th>
<th>Reference</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>server-sig-algs</td>
<td>[this document]</td>
<td>Section 3.1</td>
</tr>
<tr>
<td>client-req-ok</td>
<td>[this document]</td>
<td>Section 3.2</td>
</tr>
<tr>
<td>no-handbrake</td>
<td>[this document]</td>
<td>Section 3.3</td>
</tr>
</tbody>
</table>

5.2.1. 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


6.2. Informative References

[SSH-RSA-SHA2]
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/