Challenge-Handshake Authentication Protocol for SOCKS V5

Status of this Memo

This document is an Internet-Draft. 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."

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Abstract

This document specifies the integration of authentication based on Challenge-Handshake Authentication Protocol into SOCKS Version 5. The primary algorithm to be used is HMAC-MD5, although the framework is general enough to permit use of MD5 or other keyed hash algorithms. This document describes the message formats and protocol details of incorporating CHAP into the SOCKS V5 authentication "subnegotiation." Support is included for authentication of server to client as well as client to server.

CHAP Method Identifier

During initial SOCKS V5 negotiation, the client and server negotiate the authentication method. The METHOD for this protocol shall be X’03’.

The HMAC-MD5 Algorithm

HMAC-MD5 is defined as a new CHAP algorithm with algorithm identifier 0x85. It uses the MD5 algorithm is defined in [RFC 1321] with the HMAC construct defined in [RFC 2104] to generate responses to given challenges; unlike in the standard MD5 CHAP, the identifier octet is
ignored. Compliant implementations MUST support the HMAC-MD5
algorithm, and MAY support others.

CHAP Exchange

Subnegotiation begins after the server has selected the CHAP
authentication method.

Message Format

In general, messages exchanged consist of a version identifier and a
set of attribute-value assertions, where attributes are single octets
and values are sequences of 0-255 octets.

```
+-----+-------+------+---------+------+------+---
| VER | NAVAS | ATT1 | VAL1LEN | VAL1 | ATT2 | ...
+-----+-------+------+---------+------+------+---
|  1  |   1   |   1  |    1    | 0-255|  1   | ...
+-----+-------+------+---------+------+------+---
```

VER contains the current version of the subnegotiation, which is
X’01’. NAVAS contains the number of attribute-value assertions to
follow. Each AVA includes ATT_i, containing the attribute, VAL_iLEN,
containing the length of VAL_i, and VAL_i. In general, robust
implementations should ignore assertions with attributes they do not
understand. This provides a powerful and general mechanism for
future extensions while allowing backward compatibility.

Notationally, a single message with a set of n assertions shall be
represented as:

```
ATT_1(VAL_1), ATT_2(VAL_2), ... ATT_n(VAL_n)
```

Note that no ordering is assigned to the set of assertions: compliant
implementations must accept them in any order.

Attribute Definitions

The following attribute definitions apply to all messages:

<table>
<thead>
<tr>
<th>ATT</th>
<th>Label</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>X’00’</td>
<td>STATUS</td>
<td>0 = success</td>
</tr>
<tr>
<td>X’01’</td>
<td>TEXT-MESSAGE</td>
<td>Informational text</td>
</tr>
<tr>
<td>X’02’</td>
<td>USER-IDENTITY</td>
<td>Contains CHAP NAME</td>
</tr>
<tr>
<td>X’03’</td>
<td>CHALLENGE</td>
<td></td>
</tr>
<tr>
<td>X’04’</td>
<td>RESPONSE</td>
<td></td>
</tr>
<tr>
<td>X’05’</td>
<td>CHARSET</td>
<td>Optional character set</td>
</tr>
</tbody>
</table>

VanHeyningen                Expires July 1998                   
[Page 2]
The TEXT-MESSAGE attribute may always be included in any message. Implementations should display its value to the user if applicable; it may be used for advisory information (e.g. warnings of pending password expiration, explanations accompanying a failure.) If presenting the message to a user is not possible or not applicable, implementations may log its contents.

The CHARSET attribute provides advisory information about the character set in use; it, too, may be present in any message. Implementations may use it to guide prompting and presentation of usernames, challenges, responses and text messages. The semantics are those defined for charset parameter in MIME [RFC 1521]; if absent, a default of US-ASCII (or a superset) must be assumed.

The IDENTIFIER is used to transport the CHAP identifier when using algorithms such as MD5 which require an identifier; it is included with all messages after the algorithm negotiation when MD5 is selected.

Algorithm Negotiation

The CHAP subnegotiation begins with the client sending a message containing the CHAP algorithms it is willing to use (e.g. HMAC-MD5, MD5.) Note that compliant implementations MUST support HMAC-MD5:

```
ALGORITHMS(<algorithms>)
```

The server responds with another message of the same form containing the one algorithm to be used for this connection:

```
ALGORITHMS(<algorithm>)
```

If the server is unable or unwilling to use any of the algorithms specified by the client, it returns a message indicating failure:

```
STATUS(failure)
```

and closes the connection.

Challenge-Response Exchange

After the algorithm is negotiated, the server sends a challenge:

```
CHALLENGE(<challenge>)
```
The client sends an appropriate response:

```
USER-IDENTITY(<username>), RESPONSE(<response>)
```

And the server indicates success or failure:

```
STATUS(success|failure)
```

after which the subnegotiation terminates and, upon success, the client should proceed with its request. Upon failure, the server must close the connection.

**Mutual Authentication**

Implementations MAY support mutual authentication of client and server. A client may request mutual authentication by including another CHALLENGE along with its response:

```
USER-IDENTITY(<username>), RESPONSE(<response>),
CHALLENGE(<challenge-2>)
```

The server should then include a RESPONSE along with the STATUS message:

```
STATUS(success|failure), RESPONSE(<response-2>)
```

Finally, the client replies with a STATUS message of its own before the subnegotiation terminates

```
STATUS(success|failure)
```

Note that both negotiations employ the same identifier. Whether the same shared secret is used in both directions is outside the scope of this specification, although use of a single secret does not create the same security considerations with SOCKS5 as are present in PPP. Since servers unable or unwilling to do mutual authentication will ignore the client’s CHALLENGE, clients should handle a lack of RESPONSE gracefully and either continue or terminate the connection in accordance with their security policy.

**Security Considerations**

Challenge-response protocols are generally designed to provide protection from passive attacks such as sniffing passwords. CHAP offers limited protection from real-time active attacks.

Algorithms other than HMAC-MD5, such as MD5 as originally specified in [RFC 1994], were created without the benefit of much subsequent
research into the area of keyed hash construction. Their design is now considered weak. A variant of CHAP called MS-CHAP [MSCHAP] is known to be particularly weak.

As in all challenge-response security mechanisms, it is important that challenges be produced in a fashion an adversary cannot predict or duplicate. As with all negotiation-based security, implementations may be vulnerable to downgrade attacks. Clients and servers should refuse to operate with methods and algorithms considere insufficiently secure

In the context of a PPP connection, a CHAP challenge may be issued at any time to reconfirm the authentication. This integration permits challenges only during connection establishment and has no provision for reconfirmation.

Acknowledgements

Thanks to Dave Blob, Wei Lu, Craig Metz, and William Perry for assistance with this document.

References


Author's Address

Marc VanHeyningen
Aventail Corporation
117 S. Main St, Suite 400
Seattle, WA  98104

Phone:  +1  206  777-5600
Fax:  +1  206  777-5656
Email: marcvh@aventail.com