INTERNET-DRAFT J. Michener, D. Fritch, M. Gayman
<draft-ietf-aft-socks-maf-01.txt> Novell, Inc.
Expires 9 February 2000 9 August 1999

Multi-Authentication Framework Method for SOCKS V5

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026

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.''

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

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

To view the entire list of current Internet-Drafts, please check the ``lid abstracts.txt'' listing contained in the Internet-Drafts Shadow directories on ftp.is.co.za (Africa), ftp.nordu.net (Northern Europe), ftp.nis.garr.it (Southern Europe), munnari.oz.au (Pacific Rim), ftp.ietf.org (US East Coast), or ftp.isi.edu (US West Coast).

Abstract

SOCKS V5 [RFC 1928] provides a means to select one from among a number of authentication methods but does not provide any means for utilizing multiple authentication methods to obtain certain desired authentication properties.

MAF is a client-initiated but server-managed framework. MAF relies on a trusted Authentication Management Server (AMS) to: 1) Select the authentication methods to be invoked, 2) order the execution of methods at the client, as appropriate, and 3) assign integrity grades to the final, composite authentication after all methods that were invoked have completed.

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.

Please send comments on this document to the aft@socks.nec.com mailing

list.

1. Introduction

During an initial SOCKS V5 negotiation, the client and server negotiate an authentication method.

The METHOD value to invoke this proposed multi authentication framework (MAF) SHALL be X'08' (this value falls within the IANA assigned range indicated in RFC 1928 and was assigned by IANA to this proposed method in August 1998.)

128-bit (16-byte) UUIDs (Universally Unique Identifiers, as defined in

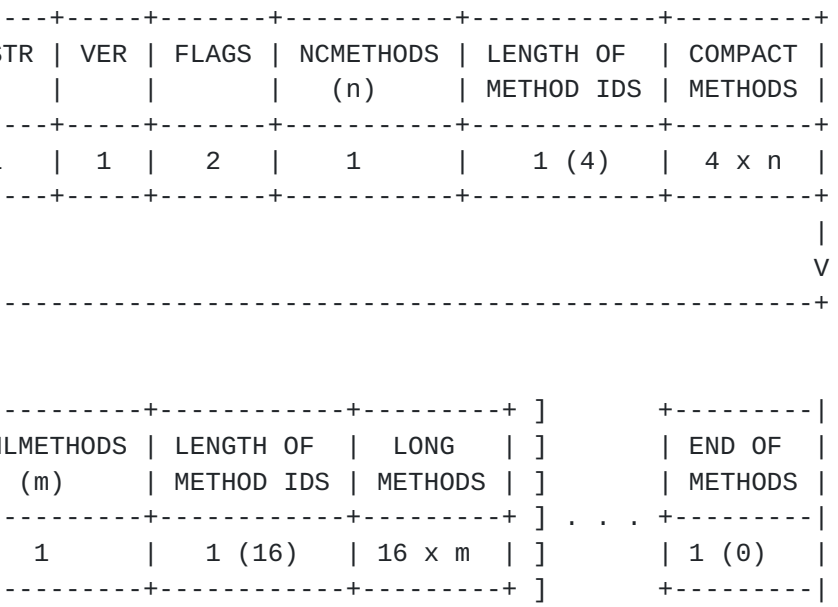
[1]) are one form of method identifier used in this protocol. GUIDs (Globally Unique Identifiers), such as those generated by development tools from Microsoft, Incorporated, are suitable as UUIDs.

Unless otherwise specified, all integer values larger than one byte will appear in the protocol messages in most-significant byte first order, i.e., network order.

2. Sub-negotiation

Sub-negotiation, as defined by the SOCKS V5 protocol, begins after the client has selected the MAF method within the SOCKS protocol. All aspects of sub-negotiation are conducted under the control of the server.

The client sends an initial MAF version identifier and potential methods list message to the server:



The INSTR field is a byte that specifies the operation being performed. The values defined at this time, and their colloquial names (in parentheses, if established), and the direction of the message (client to server, server to client, or either) are:

X'FF' Failure and disconnect (Failure), server to client
X'00' Success (Done), server to client
X'01' MAF methods supported (Can Do or List), client to server
X'02' Request additional MAF methods supported (Send More Can Do), server to client
X'03' Do, server to client
X'04' What next, client to server
X'05' Process, either

X'06' Acknowledge, either
X'07' Process OEM specific (OEM), either

To start the sub-negotiation the INSTR field is set to ``MAF methods supported'' (Can Do), X'01'.

The VER field is a byte and is set to the version of the MAF protocol. At this time VER will be X'01'.

The FLAGS field is an unsigned 16-bit value. At this time it is set to X'0000'. It provides for future tuning or extensions of the protocol.

The MAF method identifiers in this second version of the design have two possible formats: A compact or short format consisting of well known 32 bit (4-byte) unsigned integer values and a long format consisting of 128 bit (16-byte) UUIDs. Compact MAF methods IDs are fixed and unalterable after they have been registered (by the IANA or other authority). UUIDs are fixed and unalterable after they have been generated and then employed to identify a particular vendor's method. Consequentially, MAF methods do not have any version identifications per se and many common version incompatibilities are thus avoided. If a method is later found to be inadequate, the revised compact method identifier SHOULD be registered or a new UUID generated by the vendor and the replacement MAF method module SHOULD be released. A bug found in a released method MAY require a new identifier or MAY retain its former UUID or registered identifier, depending on the nature of the bug and the distribution of the method.

NCMETHODS is a byte containing the number of 32-bit compact method IDs in the COMPACT METHODS field that follows the first LENGTH OF METHOD IDS field, which has a fixed value of 4 above. NLMETHODS is the number of **16 byte UUIDs in the LONG METHODS field that follows the second LENGTH** OF METHOD IDS field, which has a fixed value of 16 above. A single byte of 0 follows the last long format method ID. The 32-bit compact method IDs are, as stated in the abstract section, in most-significant byte first order and they are not necessarily aligned on 4-byte boundaries in the packet. The byte values in the 16-byte UUIDs are in their standard order as defined by the reference above to the OSF DCE document that describes the algorithm for generating them.

If the client has more MAF method IDs, in either compact or long form, that it can send to the server, the client includes the compact method ID X'00000002' anywhere in the list of 32-bit values to notify the server that more IDs are available.

It is the prerogative of the server whether or not to ask for the additional method IDs by sending to the client a reply with a value of X'02' in the INSTR field (Send More Can Do) with a value of X'01' in VER.

The packet diagrammed above is a specific instance of the general design for this message, which allows the method IDs to be any size from 1 to **255 bytes and allows from 1 to 255 methods of the same length to be** grouped together. The three fields bracketed by [and] constitute the general structure that can be repeated as needed in the message, with

different values. The last byte of 0 (which would be the number of methods in the next grouping) is always present after the last group of method IDs. To simplify the logic of constructing this packet, all method IDs of the same length SHOULD be sent together in the message. In consideration of future method IDs of lengths other than 4 and 16, it

is likely that the length of a method ID will implicitly indicate the identification space of the value, i.e., an ID that is 4 bytes long is a registered compact format ID, an ID of 16 bytes is a long format UUID, etc. Two IDs of the same value (disregarding more significant bytes of zeroes or encoding differences) will likely not identify the same method if they are of different lengths, e.g., a future one byte ID of 0x09 will not be the same as a four byte ID of 0x00000009.

Nothing SHOULD be imputed or inferred from the order of the method IDs (either compact or long) in the data sent from the client to the server in this packet. Thus it is allowed that the compact method IDs could follow the UUIDs. Either format of IDs could also be absent from the message.

The server MAY select one of the MAF methods identified in either METHODS field (if none of the methods would meet the requirements of authentication policies on the server and the client did not indicate that more method IDs were available, the value of the method selected would be Failure) and send a Do command:

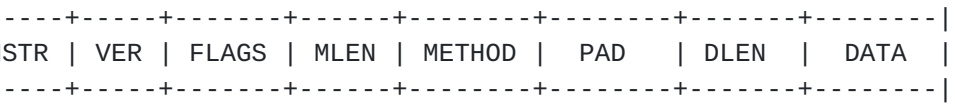
INSTR	VER	FLAGS	MLEN	METHOD
1	1	2	1	MLEN

The INSTR field is set to ``Do'', X'03'. As above, the VER field is set to the version of the MAF protocol. At this time VER is set to X'01' and the FLAGS field is set to X'0000'. The MAF method ID to be performed is entered in the METHOD field. The length of this field in bytes is the value of the single byte MLEN, either 4 for a compact method identifier or 16 for a long identifier.

If the server instructs the client to send more method IDs, via the X'02' ``Request additional MAF methods supported'' instruction, the server will use the FLAGS field to specify either a relative list (the client is to send only methods IDs that have not already been sent) or an absolute list (the client is to start sending the method IDs again as the original list, starting with the first method ID it sent). The FLAGS field will be X'0000' for a relative method ID list or X'0001' for an absolute method ID list.

The client and the server protocol managers then call the appropriate executable modules, or subroutines, to run the specified authentication method. It is anticipated that each method would be a separate binary, executable file. The mapping of method IDs, in either compact or long form, to the names and paths of the files containing the executable code is an implementation issue and is not addressed here. The exchange between the selected client and server modules will use the following

data packet:



```
|   1   |  1  |   2   |   1  |  MLEN  | 0 to 3 |   4   |  DLEN  |
|-------+-----+-------+------+--------+--------+-------+--------|
```

The INSTR field is set to ``Process'', X'05'. As above, the VER field is set to the version of the MAF protocol. At this time VER is set to X'01' and the FLAGS field is set to X'0000'. The ID of the MAF method being performed is present in the METHOD field. The length of this field in bytes is the value of the single byte MLEN, either 4 for a compact method identifier or 16 for a long identifier. The particular data being processed is sent to the client from server or from the server to the client in the DATA field as a byte array, with the length of the array specified in the DLEN field. The PAD field is 0 to 3 bytes of unspecified values to align the DLEN field to a 4-byte boundary relative to the beginning of the packet.

Exchange of data between the method on the client and the method on the server, via ``Process'' packets, continues as long as the two need to run during the particular authentication process.

The client and server methods return success or failure to their respective client and server protocol manager modules. In either outcome case, the client sends the following message to the server:

```
|-------+-----+-------|
| INSTR | VER | FLAGS |
|-------+-----+-------|
|   1   |  1  |   2   |
|-------+-----+-------|
```

The INSTR field is set to ``What next'', X'04'. At this time VER is set to X'01' and the FLAGS field is set to X'0000'.

In the event of failure of an authentication method or of the authentication process, the server MAY instruct the client to close (disconnect) the connection.

If the method succeeded, or if it failed and the server does not need to direct the client to close the connection, the server MAY instruct the client to execute another MAF method module.

At the end of the process, as determined by policies and controls on the server, the server will send the following in response to ``What next'':

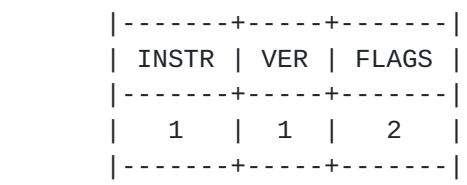
```
|-------+-----+-------+------+--------|
| INSTR | VER | FLAGS | MLEN | METHOD |
|-------+-----+-------+------+--------|
|   1   |  1  |   2   |   1  |  MLEN  |
|-------+-----+-------+------+--------|
```

If the composite authentication process succeeded, the INSTR field will

be set to ``Success'', X'00', MLEN to 4, and METHOD to Success, X'00000000'. If the authentication process failed, the INSTR field will be set to ``Failure'', X'FF', MLEN to 4, and METHOD to Failure, X'FFFFFFFF'. In either outcome case, VER is set to X'00' and FLAGS is

set to X'0000'.

Upon receipt of either the ``Success'' or ``Failure'' messages by the client, the client will send an acknowledgement to the server. The server will indicate reception of the acknowledge by replying with an acknowledge message as well:



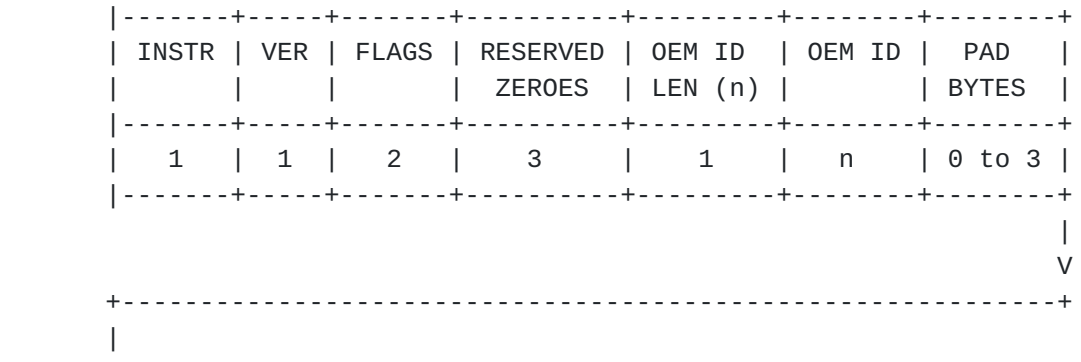
The INSTR field is set to ``Acknowledge'', X'06'. At this time VER is set to X'01' and the FLAGS field is set to X'0000'. The acknowledge message serves to synchronize the MAF protocol client and the server.

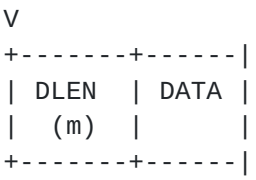
3. Process OEM Specific

The ``Process OEM specific'' instruction, X'07', provides a mechanism for the client and server MAF protocol managers to exchange data before or after a method has been invoked (when a given method is running on the client and server, the implementers are free to use the data field of the ``Process'' message to implement any form of communication between the client and the server modules.) The server can send an ``OEM'' message only in response to ``What next'', ``Can Do'', or ``OEM'' messages from the client.

The client can send an ``OEM'' message to the server before sending a ``Can Do'' or ``What next'' message or in response to a previous ``Process OEM specific'', ``Success'', or ``Failure'' message from the server. When one end sends an ``OEM'' message the other end MUST respond with an ``OEM'' message as an acknowledgment. The acknowledgment message can contain no significant data, if desired. When the exchange of ``OEM'' messages is complete, the protocol managers continue with the standard aspects of MAF.

The ``Process OEM specific'' message is composed as follows:




V
DLEN
(m)
DATA

```
|   4   |  m   |
+-------+------|
```

The INSTR field is set to ``OEM'', X'07'. As above, the VER field is set to the version of the MAF protocol. At this time VER is set to X'01' and the FLAGS field is set to X'0000'. The RESERVED ZEROES field is for future use or features. The length in bytes of the OEM ID field is entered into the OEM ID LEN field. In this version of MAF, the length will be either 4 for compact OEM IDs or 16 for long OEM IDs that are UUIDs. The particular data being processed is sent from one end to the other in the DATA field as a byte array, with the length of the array specified in the DLEN field. The PAD BYTES field is 0 to 3 bytes of unspecified values to align the DLEN field to a 4-byte boundary relative to the beginning of the packet (for the 4- or 16-byte OEM IDs proposed here, this field is absent.) For an acknowledgment message, DLEN can be zero.

If the client or server receives an ``OEM'' message with an OEM ID that it does not recognize or support, it will reply with an ``OEM'' message with the OEM ID set to X'FFFFFFFF' and DLEN set to 0 to so indicate.

3.1 Current Compact OEM IDs

X'FFFFFFFF'	OEM ID not recognized or supported
X'00000000' To X'00000003'	Internal Test IDs
X'00000004'	Reserved
X'00000005' To X'0000FFFF'	Reserved for proprietary IDs, assigned by Novell
X'00010000' and up	General IDs, assigned by the IANA

3.2 Current Compact MAF Method IDs

X'FFFFFFFF'	Failure
X'00000000'	Success
X'00000001'	Internal Test Method ID
X'00000002'	More method IDs are available
X'00000003'	Reserved
X'00000004'	Reserved

X'00000005'	
To	Reserved for proprietary method IDs, assigned
X'0000FFFF'	by Novell

X'00010000' and up	General MAF authentication method IDs, assigned by the IANA

4. Security Considerations

MAF enables the efficient combination of multiple authentication mechanisms (allowing the combination of something the user holds, something the user knows, and something the user is). This allows for the reliable establishment of user identity during the authentication session if the methods are appropriately chosen and appropriately managed. The selection of methods and their management are not addressed by MAF. Improper selection of methods and inappropriate management of the authentication process can invalidate any authentication, including that provided by MAF.

If critical data such as long-term passwords or biometric data are exchanged between the client and the server, appropriate steps SHOULD be taken to secure it at the client (so that an attacker cannot acquire this data), to secure it over the communications channel (using encryption), and to secure this data and its processing at the server. Without appropriate security and integrity at each link in the authentication process, the integrity of the authentication cannot be assured.

5. References

[RFC 1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., & Jones, L., ``SOCKS Protocol V5'', April 1996.

[1] Steven Miller, ``DEC/HP Network Computing Architecture Remote Procedure Call RunTime Extension Specification Version OSF TX1.0.11'', July 23, 1992.

6. Acknowledgements

We express our thanks to Tolga Acar, Fred Ghiradelli, Tammy Green, and Hal Henderson for assistance in the development of this document.

7. Authors' Addresses

John Michener
Novell, Inc.
122 East 1700 South
Provo Utah, 84606-6194

Phone: +1 801 861-7000
Fax: +1 801 861-2522
Email: jmichener@novell.com

Dan Fritch

Novell, Inc.
122 East 1700 South
Provo Utah, 84606-6194
Phone: +1 801 861-7000

Fax: +1 801 861-2522
Email: dfritch@novell.com

Mark Gayman
Novell, Inc.
122 East 1700 South
Provo Utah, 84606-6194

Phone: +1 801 861-7000
Fax: +1 801 861-2522
Email: mgayman@novell.com

8. Full Copyright Statement