                   Security Negotiation for WebNFS

Status of this Memo

   This memo provides information for the Internet community. It does
   not specify an Internet standard of any kind. Distribution of this
   memo is unlimited.

Copyright Notice

Abstract

   This document describes a protocol for a WebNFS client [RFC2054] to
   negotiate the desired security mechanism with a WebNFS server
   [RFC2055] before the WebNFS client falls back to the MOUNT v3
   protocol [RFC1813]. This document is provided so that people can
   write compatible implementations.

Table of Contents

1. Introduction

   The MOUNT protocol is used by an NFS client to obtain the necessary
   filehandle for data access. MOUNT versions 1 and 2 [RFC1094] return
   NFS version 2 filehandles, whereas MOUNT version 3 [RFC1813] returns
   NFS version 3 filehandles.

   Among the existing versions of the MOUNT protocol, only the MOUNT v3
   provides an RPC procedure (MOUNTPROC3_MNT) which facilitates security
   negotiation between an NFS v3 client and an NSF v3 server. When this
   RPC procedure succeeds (MNT3_OK) the server returns to the client an
   array of security mechanisms it supports for the specified pathname,
   in addition to an NFS v3 filehandle.

   A security mechanism referred to in this document is a generalized
   security flavor which can be an RPC authentication flavor [RFC1831]
   or a security flavor referred to in the RPCSEC_GSS protocol
   [RFC2203]. A security mechanism is represented as a four-octet
   integer.

   No RPC procedures are available for security negotiation in versions
   1 or 2 of the MOUNT protocol.

   The NFS mount command provides a "sec=" option for an NFS client to
   specify the desired security mechanism to use for NFS transactions.
   If this mount option is not specified, the default action is to use
   the default security mechanism over NFS v2 mounts, or to negotiate a
   security mechanism via the MOUNTPROC3_MNT procedure of MOUNT v3 and
   use it over NFS v3 mounts. In the latter, the client picks the first
   security mechanism in the array returned from the server that is also
   supported on the client.

   As specified in RFC 2054, a WebNFS client first assumes that the
   server supports WebNFS and uses the publsc filehandle as the initial
   filehandle for data access, eliminating the need for the MOUNT
   protocol. The WebNFS client falls back to MOUNT if the server does
   not support WebNFS.

   Since a WebNFS client does not use MOUNT initially, the
   MOUNTPROC3_MNT procedure of MOUNT v3 is not available for security
   negotiation until the WebNFS client falls back to MOUNT. A viable
   protocol needs to be devised for the WebNFS client to negotiate
   security mechanisms with the server in the absence of the
   MOUNTPROC3_MNT procedure.

   The WebNFS security negotiation protocol must meet the following
   requirements:

        - Must work seamlessly with NFS v2 and v3, and the WebNFS
           protocols

        - Must be backward compatible with servers that do not support
           this negotiation

        - Minimum number of network turnarounds (latency)

   This document describes the WebNFS security negotiation protocol
   developed by Sun Microsystems, Inc. Terminology and definitions from
   RFCs 2054 and 2055 are used in this document. The reader is expected
   to be familiar with them.

2. Security Negotiation Multi-component LOOKUP

   The goal of the WebNFS security negotiation is to allow a WebNFS
   client to identify a security mechanism which is used by the WebNFS
   server to protect a specified path and is also supported by the
   client. The WebNFS client initiates the negotiation by sending the
   WebNFS server the path. The WebNFS server responds with the array of
   security mechanisms it uses to secure the specified path. From the
   array of security mechanisms the WebNFS client selects the first one
   that it also supports.

   Without introducing a new WebNFS request, the WebNFS security
   negotiation is achieved by modifying the request and response of the
   existing multi-component LOOKUP (MCL) operation [RFC2055]. Note that
   the MCL operation is accomplished using the LOOKUP procedure
   (NFSPROC3_LOOKUP for NFS v3 and NFSPROC_LOOKUP for NFS v2). This and
   the next sections describe how the MCL request and response are
   modified to facilitate WebNFS security negotiation.

   For ease of reference, the modified MCL request is henceforth
   referred to as SNEGO-MCL (security negotiation multi-component
   LOOKUP) request.

   A multi-component LOOKUP request [RFC2055] is composed of a public
   filehandle and a multi-component path:

        For Canonical Path:

                LOOKUP FH=0x0, "/a/b/c"

For Native Path:

        LOOKUP FH=0x0, 0x80 "a:b:c"

A multi-component path is either an ASCII string of slash separated
components or a 0x80 character followed by a native path. Note that
a multi-component LOOKUP implies the use of the public filehandle in
the LOOKUP.

Similar to the MCL request, a SNEGO-MCL request consists of a public
filehandle and a pathname. However, the pathname is uniquely
composed, as described below, to distinguish it from other pathnames.

The pathname used in a SNEGO-MCL is the regular WebNFS multi-
component path prefixed with two octets. The first prefixed octet is
the 0x81 non-ascii character, similar to the 0x80 non-ascii character
for the native paths. This octet represents client's indication to
negotiate security mechanisms. It is followed by the security index
octet which stores the current value of the index into the array of
security mechanisms to be returned from the server. The security
index always starts with one and gets incremented as negotiation
continues. It is then followed by the pathname, either an ASCII
string of slash separated canonical components or 0x80 and a native
path.

A security negotiation multi-component LOOKUP request looks like
this:

    For Canonical Path:

        LOOKUP FH=0x0, 0x81 <sec-index> "/a/b/c"

    For Native Path:

        LOOKUP FH=0x0, 0x81 <sec-index> 0x80 "a:b:c"

In the next section we will see how the MCL response is modified for
WebNFS security negotiation.

3. Overloaded Filehandle

As described in RFC2054, if a multi-component LOOKUP request
succeeds, the server responds with a valid filehandle:

    LOOKUP FH=0x0, "a/b/c"
                   ----------->
                   <-----------
                               FH=0x3

NFS filehandles are used to uniquely identify a particular file or
directory on the server and are opaque to the client. The client
neither examines a filehandle nor has any knowledge of its contents.
Thus, filehandles make an ideal repository for the server to return
the array of security mechanisms to the client in response to a
SNEGO-MCL request.

To a successful SNEGO-MCL request the server responds, in place of
the filehandle, with an array of integers that represents the valid
security mechanisms the client must use to access the given path. A
length field is introduced to store the size (in octets) of the array
of integers.

As the filehandles are limited in size (32 octets for NFS v2 and up
to 64 octets for NFS v3), it can happen that there are more security
mechanisms than the filehandles can accommodate. To circumvent this
problem, a one-octet status field is introduced which indicates
whether there are more security mechanisms (1 means yes, 0 means no)
that require the client to perform another SNEGO-MCL to get them.

To summarize, the response to a SNEGO-MCL request contains, in place
of the filehandle, the length field, the status field, and the array
of security mechanisms:

     FH: length, status, {sec_1 sec_2 ... sec_n}

The next two sub-sections describe how NFS v2 and v3 filehandles are
"overloaded" to carry the length and status fields and the array of
security mechanisms.

3.1 Overloaded NFS Version 2 Filehandle

A regular NFS v2 filehandle is defined in RFC1094 as an opaque value
occupying 32 octets:

```
  1   2   3   4                                              32
 +---+---+---+---+---+---+---+---+    +---+---+---+---+---+---+---+
 |   |   |   |   |   |   |   |   | ...|   |   |   |   |   |   |   |
 +---+---+---+---+---+---+---+---+    +---+---+---+---+---+---+---+
```
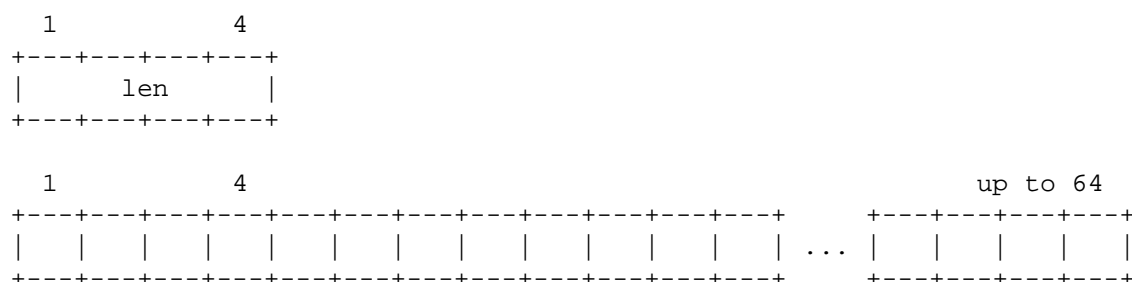
An overloaded NFS v2 filehandle looks like this:

```
  1   2   3   4   5           8                              32
 +---+---+---+---+---+---+---+---+    +---+---+---+---+    +---+---+
 | l | s |   |   |    sec_1   | ...|    sec_n   | ...|    |   |   |
 +---+---+---+---+---+---+---+---+    +---+---+---+---+    +---+---+
```
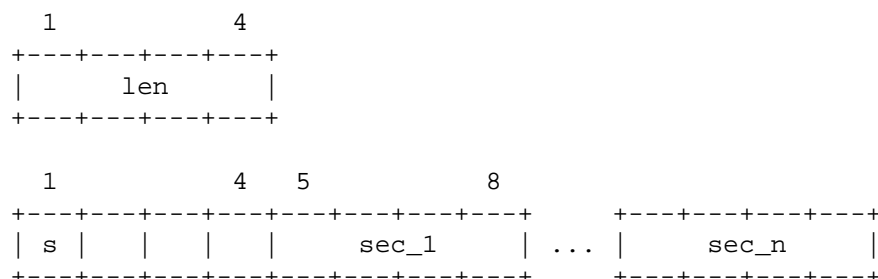
   Note that the first four octets of an overloaded NFS v2 filehandle
   contain the length octet, the status octet, and two padded octets to
   make them XDR four-octet aligned. The length octet l = 4 * n, where
   n is the number of security mechanisms sent in the current overloaded
   filehandle. Apparently, an overloaded NFS v2 filehandle can carry up
   to seven security mechanisms.

3.2 Overloaded NFS Version 3 Filehandle

   A regular NFS v3 filehandle is defined in RFC1813 as a variable
   length opaque value occupying up to 64 octets. The length of the
   filehandle is indicated by an integer value contained in a four octet
   value which describes the number of valid octets that follow:

```
   1           4
+---+---+---+---+
|      len      |
+---+---+---+---+


   1           4                                               up to 64
+---+---+---+---+---+---+---+---+---+---+---+---+     +---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   | ... |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+     +---+---+---+---+
```

An overloaded NFS v3 filehandle looks like the following:

```
   1           4
+---+---+---+---+
|      len      |
+---+---+---+---+


   1           4   5               8
+---+---+---+---+---+---+---+---+     +---+---+---+---+
| s |   |   |   |      sec_1    | ... |      sec_n    |
+---+---+---+---+---+---+---+---+     +---+---+---+---+
```

   Here, len = 4 * (n+1). Again, n is the number of security mechanisms
   contained in the current overloaded filehandle. Three octets are
   padded after the status octet to meet the XDR four-octet alignment
   requirement. An overloaded NFS v3 filehandle can carry up to fifteen
   security mechanisms.

4. WebNFS Security Negotiation

   With the SNEGO-MCL request and the overloaded NFS v2 and v3
   filehandles defined above, the following diagram depicts the WebNFS
   security negotiation protocol:

```
   Client                                          Server
   ------                                          ------

      LOOKUP FH=0x0, 0x81 <sec-index> "path"
                      ----------->
                      <-----------
                          FH: length, status, {sec_1  sec_2 ... sec_n}
```

   where
      0x81 represents client's indication to negotiate security
      mechanisms with the server,

      path is either an ASCII string of slash separated components or
      0x80 and a native path,

      sec-index, one octet, contains the index into the array of
      security mechanisms the server uses to protect the specified path,

      status, one octet, indicates whether there are more security
      mechanisms (1 means yes, 0 means no) that require the client to
      perform another SNEGO-MCL to get them,

      length (one octet for NFS v2 and four octets for NFS v3) describes
      the number of valid octets that follow,

      {sec_1 sec_2 ... sec_n} represents the array of security
      mechanisms. As noted earlier, each security mechanism is
      represented by a four-octet integer.

   Here is an example showing the WebNFS security negotiation protocol
   with NFS v2. In the example it is assumed the server shares /export
   with 10 security mechanisms {0x3900 0x3901 0x3902 ... 0x3909} on the
   export, two SNEGO-MCL requests would be needed for the client to get
   the complete security information:

```
    LOOKUP FH=0x0, 0x81 0x01 "/export"
                      ----------->
                      <-----------
        0x1c, 0x01, {0x3900 0x3901 0x3902 0x3903 0x3904 0x3905 0x3906}

    LOOKUP FH=0x0, 0x81 0x08 "/export"
                      ----------->
                      <-----------
        0x0c, 0x00, {0x3907 0x3908 0x3909}
```

The order of the security mechanisms returned in an overloaded
filehandle implies preferences, i.e., one is more recommended than
those following it. The ordering is the same as that returned by the
MOUNT v3 protocol.

The following shows a typical scenario which illustrates how the
WebNFS security negotiation is accomplished in the course of
accessing publicly shared filesystems.

Normally, a WebNFS client first makes a regular multi-component
LOOKUP request using the public filehandle to obtain the filehandle
for the specified path. Since the WebNFS client does not have any
prior knowledge as to how the path is protected by the server the
default security mechanism is used in this first multi-component
LOOKUP. If the default security mechanism does not meet server's
requirements, the server replies with the AUTH_TOOWEAK RPC
authentication error, indicating that the default security mechanism
is not valid and the WebNFS client needs to use a stronger one.

Upon receiving the AUTH_TOOWEAK error, to find out what security
mechanisms are required to access the specified path the WebNFS
client sends a SNEGO-qMCL request, using the default security
mechanism.

If the SNEGO-MCL request succeeds the server responds with the
filehandle overloaded with the array of security mechanisms required
for the specified path. If the server does not support WebNFS
security negotiation, the SNEGO-MCL request fails with NFSERR_IO for
NFS v2 or NFS3ERR_IO for NFS v3 [RFC2055].

Depending on the size of the array of security mechanisms, the WebNFS
client may have to make more SNEGO-MCL requests to get the complete
array.

For successful SNEGO-MCL requests, the WebNFS client retrieves the
array of security mechanisms from the overloaded filehandle, selects
an appropriate one, and issues a regular multi-component LOOKUP using
the selected security mechanism to acquire the filehandle.

All subsequent NFS requests are then made using the selected security
mechanism and the filehandle.

The following depicts the scenario outlined above. It is assumed
that the server shares /export/home as follows:

        share -o sec=sec_1:sec_2:sec_3,public /export/home

   and AUTH_SYS is the client's default security mechanism and is not
   one of {sec_1, sec_2, sec_3}.

```
        Client                                          Server
        ------                                          ------

        LOOKUP FH=0x0, "/export/home"
                               AUTH_SYS
                               ----------->
                               <-----------
                                                        AUTH_TOOWEAK

        LOOKUP FH=0x0, 0x81 0x01 "/export/home"
                               AUTH_SYS
                               ----------->
                               <-----------
                    overloaded FH: length, status, {sec_1 sec_2 sec_3}

        LOOKUP FH=0x0, "/export/home"
                                 sec_n
                               ----------->
                               <-----------
                                                        FH = 0x01

        NFS request with FH=0x01
                                 sec_n
                               ----------->
                               <-----------
                                                        ...
```

   In the above scenario, the first request is a regular multi-component
   LOOKUP which fails with the AUTH_TOOWEAK error. The client then
   issues a SNEGO-MCL request to get the security information.

   There are WebNFS implementations that allow the public filehandle to
   work with NFS protocol procedures other than LOOKUP. For those
   WebNFS implementations, if the first request is not a regular multi-
   component LOOKUP and it fails with AUTH_TOOWEAK, the client should
   issue a SNEGO-MCL with

        0x81 0x01 "."

   as the path to get the security information.

5. Security Considerations

   The reader may note that no mandatory security mechanisms are
   specified in the protocol that the client must use in making SNEGO-
   MCL requests. Normally, the client uses the default security
   mechanism configured on his system in the first SNEGO-MCL request.
   If the default security mechanism is not valid the server replies
   with the AUTH_TOOWEAK error. In this case the server does not return
   the array of security mechanisms to the client. The client can then
   make another SNEGO-MCL request using a stronger security mechanism.
   This continues until the client hits a valid one or has exhausted all
   the supported security mechanisms.

6. References

   [RFC1094] Sun Microsystems, Inc., "NFS: Network File System Protocol
             Specification", RFC 1094, March 1989.
             http://www.ietf.org/rfc/rfc1094.txt

   [RFC1813] Callaghan, B., Pawlowski, B. and P. Staubach, "NFS Version
             3 Protocol Specification", RFC 1813, June 1995.
             http://www.ietf.org/rfc/rfc1813.txt

   [RFC2054] Callaghan, B., "WebNFS Client Specification", RFC 2054,
             October 1996. http://www.ietf.org/rfc/rfc2054.txt

   [RFC2055] Callaghan, B., "WebNFS Server Specification", RFC 2055,
             October 1996. http://www.ietf.org/rfc/rfc2055.txt

   [RFC2203] Eisler, M., Chiu, A. and Ling, L., "RPCSEC_GSS Protocol
             Specification", RFC 2203, September 1997.
             http://www.ietf.org/rfc/rfc2203.txt

7. Acknowledgements

   This specification was extensively brainstormed and reviewed by the
   NFS group of Solaris Software Division.

8. Authors' Addresses

   Alex Chiu
   Sun Microsystems, Inc.
   901 San Antonio Road
   Palo Alto, CA 94303

   Phone: +1 (650) 786-6465
   EMail: alex.chiu@Eng.sun.com

   Mike Eisler
   Sun Microsystems, Inc.
   901 San Antonio Road
   Palo Alto, CA 94303

   Phone: +1 (719) 599-9026
   EMail: michael.eisler@Eng.sun.com

   Brent Callaghan
   Sun Microsystems, Inc.
   901 San Antonio Road
   Palo Alto, CA 94303

   Phone: +1 (650) 786-5067
   EMail: brent.callaghan@Eng.sun.com

9. Full Copyright Statement

Acknowledgement