NFSv4
Internet-Draft
Intended status: Standards Track
Expires: June 16, 2016

W. Adamson
NetApp
N. Williams
Cryptonector

December 14, 2015

Remote Procedure Call (RPC) Security Version 3
draft-ietf-nfsv4-rpcsec-gssv3-14

Abstract

This document specifies version 3 of the Remote Procedure Call (RPC) security protocol (RPCSEC_GSS). This protocol provides support for multi-principal authentication of client hosts and user principals to a server (constructed by generic composition), security label assertions for multi-level and type enforcement, structured privilege assertions, and channel bindings.

Requirements Language

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 [RFC2119].

Status of This Memo

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on June 16, 2016.

Copyright Notice

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.

Table of Contents

1. Introduction and Motivation 2
  1.1. Added Functionality 4
  1.2. XDR Code Extraction 5
2. The RPCSEC_GSSv3 Protocol 5
  2.1. Compatibility with RPCSEC_GSSv2 6
  2.2. Version Negotiation 6
  2.3. New REPLY Verifier 6
  2.4. XDR Code Preliminaries 7
  2.5. RPCSEC_GSS_BIND_CHANNEL Operation 9
  2.6. New auth_stat Values 9
  2.7. New Control Procedures 10
    2.7.1. New Control Procedure - RPCSEC_GSS_CREATE 10
    2.7.2. New Control Procedure - RPCSEC_GSS_LIST 17
  2.8. Extensibility 18
3. Operational Recommendation for Deployment 19
4. Security Considerations 19
5. IANA Considerations 20
6. References 20
  6.1. Normative References 20
  6.2. Informative References 20
Appendix A. Acknowledgments 21
Appendix B. RFC Editor Notes 21
Authors' Addresses 21

1. Introduction and Motivation

The original RPCSEC_GSS protocol [RFC2203] provided for authentication of RPC clients and servers to each other using the Generic Security Services Application Programming Interface (GSS-API) [RFC2743]. The second version of RPCSEC_GSS [RFC5403] added support for channel bindings [RFC5056].

Existing GSS-API mechanisms are insufficient for communicating certain aspects of authority to a server. The GSS-API and its mechanisms certainly could be extended to address this shortcoming.

However, here it is addressed at the application layer, i.e., in RPCSEC_GSS.

A major motivation for version 3 RPCSEC_GSS (RPCSEC_GSSv3) is to add support for multi-level (labeled) security and server-side copy for NFSv4.

Multi-Level Security (MLS) is a traditional model where subjects (processes) are given a security level (Unclassified, Secret, Top Secret, etc.) and objects (files) are given security labels that mandate the access of the subject to the object (see [BL73] and [RFC4301]).

Labeled NFS (see Section 9 of [NFSv4.2]) uses an MLS policy with Mandatory Access Control (MAC) systems as defined in [RFC4949]. Labeled NFS stores MAC file object labels on the NFS server and enables client Guest Mode MAC as described in Section 4.3 of [RFC7204]. RPCSEC_GSSv3 label assertions assert client MAC process subject labels to enable Full Mode MAC when combined with Labeled NFS as described in Section 3.3 of [RFC7204].

A traditional inter-server file copy entails the user gaining access to a file on the source, reading it, and writing it to a file on the destination. In secure NFSv4 inter-server server-side copy (see Section 4 of [NFSv4.2]), the user first secures access to both source and destination files, and then uses NFSv4.2 defined RPCSEC_GSSv3 structured privileges to authorize the destination to copy the file from the source on behalf of the user.

Multi-principal assertions can be used to address shared cache poisoning attacks (see Section 9 of [AFS-RXGK]) on the client cache by a user. As described in Section 7 of [AFS-RXGK], multi-user machines with a single cache manager can fetch and cache data on a users' behalf, and re-display it for another user from the cache without re-fetching the data from the server. The initial data acquisition is authenticated by the first user's credentials, and if only that user's credentials are used, it may be possible for a malicious user or users to "poison" the cache for other users by introducing bogus data into the cache.

Another use of the multi-principal assertion is the secure conveyance of privilege information for processes running with more (or even with less) privilege than the user normally would be accorded.

1.1. Added Functionality

We therefore describe RPCSEC_GSS version 3 (RPCSEC_GSSv3). RPCSEC_GSSv3 is the same as RPCSEC_GSSv2 [RFC5403], except that the following assertions of authority have been added.

- Security labels for Full Mode security type enforcement, and other labeled security models (See Section 5.5.1 in [RFC7204]).

- Application-specific structured privileges. For an example see server-side copy [NFSv4.2].

- Multi-principal authentication of the client host and user to the server done by binding two RPCSEC_GSS handles.

- Simplified channel binding.

Assertions of labels and privileges are evaluated by the server, which may then map the asserted values to other values, all according to server-side policy. See [NFSv4.2].

An option for enumerating server supported label format specifiers (LFS) is provided. See Section 2 and Section 3.3 in [RFC7204] for detail.

Note that there is no RPCSEC_GSS_CREATE payload that is REQUIRED to implement. RPCSEC_GSSv3 implementations are feature driven. Besides implementing the RPCSEC_GSS_CREATE operation and payloads for the desired features, all RPCSEC_GSSv3 implementation MUST implement:

- The new GSS version number (Section 2.2).

- The new reply verifier (Section 2.3).

- The new auth stat values (Section 2.6).

RPCSEC_GSSv3 targets implementing a desired feature must also implement the RPCSEC_GSS_LIST operation, and the RPCSEC_GSS_CREATE operation replies for unsupported features.

- For multi-principal authentication (Section 2.7.1.1), the target indicates no support by not including a rgss3_gss_mp_auth value in the rgss3_create_res.

- For channel bindings (Section 2.7.1.2) the target indicates no support by not including a rgss3_chan_binding value in the rgss3_create_res.

- o For label assertions the target indicates no support by returning the new RPCSEC_GSS_LABEL_PROBLEM auth stat (See Section 2.7.1.3).
- o For structured privilege assertions the target indicates no support by returning the new RPCSEC_GSS_UNKNOWN_MESSAGE auth stat (See Section 2.7.1.4).

1.2. XDR Code Extraction

This document contains the External Data Representation (XDR) ([RFC4506]) definitions for the RPCSEC_GSSv3 protocol. The XDR description is provided in this document in a way that makes it simple for the reader to extract into ready to compile form. The reader can feed this document in the following shell script to produce the machine readable XDR description of RPCSEC_GSSv3:

```
<CODE BEGINS>

#!/bin/sh
grep "^  *///" | sed 's?^  */// ??' | sed 's?^  *///$??'

<CODE ENDS>
```

I.e. if the above script is stored in a file called "extract.sh", and this document is in a file called "spec.txt", then the reader can do:

```
<CODE BEGINS>

 sh extract.sh < spec.txt > rpcsec_gss_v3.x

<CODE ENDS>
```

The effect of the script is to remove leading white space from each line, plus a sentinel sequence of "///".

2. The RPCSEC_GSSv3 Protocol

RPCSEC_GSS version 3 (RPCSEC_GSSv3) is very similar to RPCSEC_GSS version 2 (RPCSEC_GSSv2) [RFC5403]. The differences are the addition of support for assertions and channel bindings are supported via a different mechanism.

The entire RPCSEC_GSSv3 protocol is not presented here. Only the differences between it and RPCSEC_GSSv2 are shown.

The use of RPCSEC_GSSv3 is structured as follows:

- A client uses an existing RPCSEC_GSSv3 context handle established in the usual manner (See Section 5.2 [RFC2203]) to protect RPCSEC_GSSv3 exchanges, this will be termed the "parent" handle.

- The server issues a "child" RPCSEC_GSSv3 handle in the RPCSEC_GSS_CREATE response which uses the underlying GSS-API security context of the parent handle in all subsequent exchanges that uses the child handle.

- An RPCSEC_GSSv3 child handle MUST NOT be used as the parent handle in an RPCSEC_GSS3_CREATE control message.

2.1. Compatibility with RPCSEC_GSSv2

The functionality of RPCSEC_GSSv2 [RFC5403] is fully supported by RPCSEC_GSSv3 with the exception of the RPCSEC_GSS_BIND_CHANNEL operation which is not supported when RPCSEC_GSSv3 is in use (see Section 2.5).

2.2. Version Negotiation

An initiator that supports version 3 of RPCSEC_GSS simply issues an RPCSEC_GSS request with the rgc_version field set to RPCSEC_GSS_VERS_3. If the target does not recognize RPCSEC_GSS_VERS_3, the target will return an RPC error per Section 5.1 of [RFC2203].

The initiator MUST NOT attempt to use an RPCSEC_GSS handle returned by version 3 of a target with version 1 or version 2 of the same target. The initiator MUST NOT attempt to use an RPCSEC_GSS handle returned by version 1 or version 2 of a target with version 3 of the same target.

2.3. New REPLY Verifier

A new reply verifier is needed for RPCSEC_GSSv3 because of a situation that arises from the use of the same GSS context by child and parent handles. Because the RPCSEC_GSSv3 child handle uses the same GSS context as the parent handle, a child and parent RPCSEC_GSSv3 handle could have the same RPCSEC_GSS sequence numbers. Since the reply verifier of previous versions of RPCSEC_GSS computes a Message Integrity Code (MIC) on just the sequence number, this provides opportunities for man in the middle attacks.

This issue is addressed in RPCSEC_GSS version 3 by computing the verifier using the exact same input as is used to compute the request verifier, except that the mtype is changed from CALL to REPLY. The

new reply verifier computes a MIC over the following RPC reply header data:

```
unsigned int xid;
msg_type mtype;    /* set to REPLY */
unsigned int rpcvers;
unsigned int prog;
unsigned int vers;
unsigned int proc;
opaque_auth  cred; /* captures the RPCSEC_GSS handle */
```

2.4. XDR Code Preliminaries

The following code fragment replaces the corresponding preliminary code shown in Figure 1 of [RFC5403]. The values in the code fragment in Section 2.6 are additions to the auth_stat enumeration. Subsequent code fragments are additions to the code for version 2 that support the new procedures defined in version 3.

<CODE BEGINS>

```
///  /*
///   * Copyright (c) 2013 IETF Trust and the persons
///   * identified as the document authors. All rights
///   * reserved.
///   *
///   * The document authors are identified in [RFC2203],
///   * [RFC5403], and [RFCTBD].
///   *
///   * Redistribution and use in source and binary forms,
///   * with or without modification, are permitted
///   * provided that the following conditions are met:
///   *
///   * o Redistributions of source code must retain the above
///   *   copyright notice, this list of conditions and the
///   *   following disclaimer.
///   *
///   * o Redistributions in binary form must reproduce the
///   *   above copyright notice, this list of
///   *   conditions and the following disclaimer in
///   *   the documentation and/or other materials
///   *   provided with the distribution.
///   *
///   * o Neither the name of Internet Society, IETF or IETF
///   *   Trust, nor the names of specific contributors, may be
///   *   used to endorse or promote products derived from this
///   *   software without specific prior written permission.
///   *
```

```
   ///   *   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS
   ///   *   AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
   ///   *   WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
   ///   *   IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
   ///   *   FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO
   ///   *   EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
   ///   *   LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
   ///   *   EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
   ///   *   NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
   ///   *   SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
   ///   *   INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
   ///   *   LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
   ///   *   OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
   ///   *   IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
   ///   *   ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
   ///   */
   ///
   ///  /*
   ///   * This code was derived from RFC2203, RFC5403, and RFCTBD.
   ///   * Please reproduce this note if possible.
   ///   */
   ///
   ///  enum rpc_gss_service_t {
   ///          /* Note: the enumerated value for 0 is reserved. */
   ///          rpc_gss_svc_none         = 1,
   ///          rpc_gss_svc_integrity    = 2,
   ///          rpc_gss_svc_privacy      = 3,
   ///          rpc_gss_svc_channel_prot = 4
   ///  };
   ///
   ///  enum rpc_gss_proc_t {
   ///           RPCSEC_GSS_DATA          = 0,
   ///           RPCSEC_GSS_INIT          = 1,
   ///           RPCSEC_GSS_CONTINUE_INIT = 2,
   ///           RPCSEC_GSS_DESTROY       = 3,
   ///           RPCSEC_GSS_BIND_CHANNEL  = 4, /* not used */
   ///           RPCSEC_GSS_CREATE        = 5, /* new */
   ///           RPCSEC_GSS_LIST          = 6  /* new */
   ///  };
   ///
   ///  struct rpc_gss_cred_vers_1_t {
   ///          rpc_gss_proc_t    gss_proc; /* control procedure */
   ///          unsigned int      seq_num;  /* sequence number */
   ///          rpc_gss_service_t service;  /* service used */
   ///          opaque            handle<>; /* context handle */
   ///  };
   ///
   ///  const RPCSEC_GSS_VERS_1 = 1;
```

```
   ///  const RPCSEC_GSS_VERS_2 = 2;
   ///  const RPCSEC_GSS_VERS_3 = 3; /* new */
   ///
   ///  union rpc_gss_cred_t switch (unsigned int rgc_version) {
   ///  case RPCSEC_GSS_VERS_1:
   ///  case RPCSEC_GSS_VERS_2:
   ///  case RPCSEC_GSS_VERS_3: /* new */
   ///          rpc_gss_cred_vers_1_t rgc_cred_v1;
   ///  };
   ///
```

<CODE ENDS>

As seen above, the RPCSEC_GSSv3 credential has the same format as the RPCSEC_GSSv1 [RFC2203] and RPCSEC_GSSv2 [RFC5403] credential. Setting the rgc_version field to 3 indicates that the initiator and target support the new RPCSEC_GSSv3 control procedures.

2.5. RPCSEC_GSS_BIND_CHANNEL Operation

RPCSEC_GSSv3 provides a channel binding assertion that replaces the RPCSEC_GSSv2 RPCSEC_GSS_BIND_CHANNEL operation.

The RPCSEC_GSS_BIND_CHANNEL operation is not supported on RPCSEC_GSS version 3 handles. If a server receives an RPCSEC_GSS_BIND_CHANNEL operation on an RPCSEC_GSSv3 handle, it MUST return a reply status of MSG_ACCEPTED with an acccept stat of PROC_UNAVAIL.

2.6. New auth_stat Values

RPCSEC_GSSv3 requires the addition of several values to the auth_stat enumerated type definition. The use of these new auth_stat values is explained throughout this document.

```
           enum auth_stat {
                   ...
                   /*
                    * RPCSEC_GSSv3 errors
                    */
                   RPCSEC_GSS_INNER_CREDPROBLEM = 15,
                   RPCSEC_GSS_LABEL_PROBLEM = 16,
                   RPCSEC_GSS_PRIVILEGE_PROBLEM = 17,
                   RPCSEC_GSS_UNKNOWN_MESSAGE = 18
           };
```

2.7. New Control Procedures

There are two new RPCSEC_GSSv3 control procedures: RPCSEC_GSS_CREATE, RPCSEC_GSS_LIST.

The RPCSEC_GSS_CREATE procedure binds any combination of assertions: multi-principal authentication, labels, structured privileges, or channel bindings to a new RPCSEC_GSSv3 context returned in the rgss3_create_res rcr_handle field.

The RPCSEC_GSS_LIST procedure queries the target for supported assertions.

RPCSEC_GSS version 3 control messages are similar to the RPCSEC_GSS version 1 and version 2 RPCSEC_GSS_DESTROY control message (see section 5.4 [RFC2203]) in that the sequence number in the request must be valid, and the header checksum in the verifier must be valid. As in RPCSEC_GSS version 1 and version 2, the RPCSEC_GSSv version 3 control messages may contain call data following the verifier in the body of the NULLPROC procedure. In other words, they look a lot like an RPCSEC_GSS data message with the header procedure set to NULLPROC.

The client MUST use one of the following security services to protect the RPCSEC_GSS_CREATE or RPCSEC_GSS_LIST control message:

- rpc_gss_svc_integrity

- rpc_gss_svc_privacy

Specifically the client MUST NOT use rpc_gss_svc_none.

RPCSEC_GSS_LIST can also use rpc_gss_svc_channel_prot (see RPCSEC_GSSv2 [RFC5403]) if the request is sent using an RPCSEC_GSSv3 child handle with channel bindings enabled as described in Section 2.7.1.2.

2.7.1. New Control Procedure - RPCSEC_GSS_CREATE

```
<CODE BEGINS>
```

```
   ///  struct rgss3_create_args {
   ///          rgss3_gss_mp_auth    *rca_mp_auth;
   ///          rgss3_chan_binding   *rca_chan_bind_mic;
   ///          rgss3_assertion_u     rca_assertions<>;
   ///  };
   ///
   ///  struct rgss3_create_res {
   ///          opaque               rcr_handle<>;
   ///          rgss3_gss_mp_auth    *rcr_mp_auth;
   ///          rgss3_chan_binding   *rcr_chan_bind_mic;
   ///          rgss3_assertion_u     rcr_assertions<>;
   ///  };
   ///
   ///  enum rgss3_assertion_type {
   ///          LABEL = 0,
   ///          PRIVS = 1
   ///  };
   ///
   ///  union rgss3_assertion_u
   ///        switch (rgss3_assertion_type atype) {
   ///  case LABEL:
   ///          rgss3_label  rau_label;
   ///  case PRIVS:
   ///          rgss3_privs  rau_privs;
   ///  default:
   ///          opaque       rau_ext<>;
   ///  };
   ///

<CODE ENDS>
```

The call data for an RPCSEC_GSS_CREATE request consists of an rgss3_create_args which binds one or more items of several kinds to the returned rcr_handle RPCSEC_GSSv3 context handle called the "child" handle:

- Multi-principal authentication: another RPCSEC_GSS context handle

- A channel binding

- Authorization assertions: labels and or privileges

The reply to this message consists of either an error or an rgss3_create_res structure. As noted in Section 2.7.1.3 and Section 2.7.1.4 successful rgss3_assertions are enumerated in rcr_assertions, and are REQUIRED be enumerated in the same order as they appeared in the rca_assertions argument.

Upon successful RPCSEC_GSS_CREATE, both the client and the server need to associate the resultant child rcr_handle context handle with the parent context handle in their GSS context caches so as to be able to reference the parent context given the child context handle.

RPCSEC_GSSv3 child handles MUST be destroyed upon the destruction of the associated parent handle.

Server implementation and policy MAY result in labels, privileges, and identities being mapped to concepts and values that are local to the server. Server policies should take into account the identity of the client and/or user as authenticated via the GSS-API.

2.7.1.1. Multi-principal Authentication

```
<CODE BEGINS>

   ///
   ///  struct rgss3_gss_mp_auth {
   ///          opaque          rgmp_handle<>; /* inner handle */
   ///          opaque          rgmp_rpcheader_mic<>;
   ///  };
   ///

<CODE ENDS>
```

RPCSEC_GSSv3 clients MAY assert a multi-principal authentication of the RPC client host principal and a user principal. This feature is needed, for example, when an RPC client host wishes to use authority assertions that the server may only grant if a user and an RPC client host are authenticated together to the server. Thus a server may refuse to grant requested authority to a user acting alone (e.g., via an unprivileged user-space program), or to an RPC client host acting alone (e.g., when an RPC client host is acting on behalf of a user) but may grant requested authority to an RPC client host acting on behalf of a user if the server identifies the user and trusts the RPC client host.

It is assumed that an unprivileged user-space program would not have access to RPC client host credentials needed to establish a GSS-API security context authenticating the RPC client host to the server, therefore an unprivileged user-space program could not create an RPCSEC_GSSv3 RPCSEC_GSS_CREATE message that successfully binds an RPC client host and a user security context.

In addition to the parent handle (Section 2), the multi-principal authentication call data has an RPCSEC_GSS version 3 handle referenced via the rgmp_handle field termed the "inner" handle.

Clients using RPCSEC_GSSv3 multi-principal authentication MUST use an RPCSEC_GSSv3 context handle that corresponds to a GSS-API security context that authenticates the RPC client host for the parent handle. For the inner context handle with RPCSEC_GSSv3 it MUST use a context handle to authenticate a user. The reverse (parent handle authenticates user, inner authenticates an RPC client host) MUST NOT be used. Other multi-principal parent and inner context handle uses might eventually make sense, but would need to be introduced in a new revision of the RPCSEC_GSS protocol.

The child context handle returned by a successful multi-principal assertion binds the inner RPCSEC_GSSv3 context handle to the parent RPCSEC_GSS context and MUST be treated by servers as authenticating the GSS-API initiator principal authenticated by the inner context handle's GSS-API security context. This principal may be mapped to a server-side notion of user or principal.

Multi-principal binding is done by including an assertion of type rgss3_gss_mp_auth in the RPCSEC_GSS_CREATE rgss3_create_args call data. The inner context handle is placed in the rgmp_handle field. A MIC of the RPC call header up to and including the credential is computed using the GSS-API security context associated with the inner context handle and is placed in rgmp_rpcheader_mic field.

The target verifies the multi-principal authentication by first confirming that the parent context used is an RPC client host context, and then verifies the rgmp_rpcheader_mic using the GSS-API security context associated with the rgmp_handle field.

On a successful verification, the rgss3_gss_mp_auth field in the rgss3_create_res reply MUST be filled in with the inner RPCSEC_GSSv3 context handle as the rgmp_handle, and a MIC computed over the RPC reply header (see section Section 2.3) using the GSS-API security context associated with the inner handle.

On failure, the rgss3_gss_mp_auth field is not sent (rgss3_gss_mp_auth is an optional field). A MSG_DENIED reply to the RPCSEC_GSS_CREATE call is formulated as usual.

As described in Section 5.3.3.3 of [RFC2203] the server maintains a list of contexts for the clients that are currently in session with it. When a client request comes in, there may not be a context corresponding to its handle. When this occurs on an RPCSEC_GSS3_CREATE request processing of the parent handle, the server rejects the request with a reply status of MSG_DENIED with the reject_stat of AUTH_ERROR and with an auth_stat value of RPCSEC_GSS_CREDPROBLEM.

A new value, RPCSEC_GSS_INNER_CREDPROBLEM, has been added to the auth_stat type. With a multi-pricipal authorization request, the server must also have a context corresponding to the inner context handle. When the server does not have a context handle corresponding to the inner context handle of a multi-pricipal authorization request, the server sends a reply status of MSG_DENIED with the reject_stat of AUTH_ERROR and with an auth_stat value of RPCSEC_GSS_INNER_CREDPROBLEM.

When processing the multi-principal authentication request, if the GSS_VerifyMIC() call on the rgmp_rpcheader_mic fails to return GSS_S_COMPLETE, the server sends a reply status of MSG_DENIED with the reject_stat of AUTH_ERROR and with an auth_stat value of RPCSEC_GSS_INNER_CREDPROBLEM.

2.7.1.2. Channel Binding

<CODE BEGINS>

```
///
///  typedef opaque rgss3_chan_binding<>;
///
```

<CODE ENDS>

RPCSEC_GSSv3 provides a different way to do channel binding than RPCSEC_GSSv2 [RFC5403]. Specifically:

a. RPCSEC_GSSv3 builds on RPCSEC_GSSv1 by reusing existing, established context handles rather than providing a different RPC security flavor for establishing context handles,

b. channel bindings data are not hashed because there is now general agreement that it is the secure channel's responsibility to produce channel bindings data of manageable size.

(a) is useful in keeping RPCSEC_GSSv3 simple in general, not just for channel binding. (b) is useful in keeping RPCSEC_GSSv3 simple specifically for channel binding.

Channel binding is accomplished as follows. The client prefixes the channel bindings data octet string with the channel type as described in [RFC5056], then the client calls GSS_GetMIC() to get a MIC of resulting octet string, using the parent RPCSEC_GSSv3 context handle's GSS-API security context. The MIC is then placed in the rca_chan_bind_mic field of RPCSEC_GSS_CREATE arguments (rgss3_create_args).

If the rca_chan_bind_mic field of the arguments of a RPCSEC_GSS_CREATE control message is set, then the server MUST verify the client's channel binding MIC if the server supports this feature. If channel binding verification succeeds then the server MUST generate a new MIC of the same channel bindings and place it in the rcr_chan_bind_mic field of the RPCSEC_GSS_CREATE rgss3_create_res results. If channel binding verification fails or the server doesn't support channel binding then the server MUST indicate this in its reply by not including a rgss3_chan_binding value in rgss3_create_res (rgss3_chan_binding is an optional field).

The client MUST verify the result's rcr_chan_bind_mic value by calling GSS_VerifyMIC() with the given MIC and the channel bindings data (including the channel type prefix). If client-side channel binding verification fails then the client MUST call RPCSEC_GSS_DESTROY. If the client requested channel binding but the server did not include an rcr_chan_binding_mic field in the results, then the client MAY continue to use the resulting context handle as though channel binding had never been requested. If the client considers channel binding critical, it MUST call RPCSEC_GSS_DESTROY.

As per-RPCSEC_GSSv2 [RFC5403]:

> "Once a successful [channel binding] procedure has been performed on an [RPCSEC_GSSv3] context handle, the initiator's implementation may map application requests for rpc_gss_svc_none and rpc_gss_svc_integrity to rpc_gss_svc_channel_prot credentials. And if the secure channel has privacy enabled, requests for rpc_gss_svc_privacy can also be mapped to rpc_gss_svc_channel_prot."

Any RPCSEC_GSSv3 child context handle that has been bound to a secure channel in this way SHOULD be used only with the rpc_gss_svc_channel_prot, and SHOULD NOT be used with rpc_gss_svc_none nor rpc_gss_svc_integrity -- if the secure channel does not provide privacy protection then the client MAY use rpc_gss_svc_privacy where privacy protection is needed or desired.

2.7.1.3. **Label Assertions**

<CODE BEGINS>

```
   ///  struct rgss3_label {
   ///          rgss3_lfs       rl_lfs;
   ///          opaque          rl_label<>;
   ///  };
   ///
   ///  struct rgss3_lfs {
   ///          unsigned int rlf_lfs_id;
   ///          unsigned int rlf_pi_id;
   ///  };
   ///

<CODE ENDS>
```

The client discovers which label format specifiers (LFS) the server supports via the RPCSEC_GSS_LIST control message. Full mode MAC is enabled when an RPCSEC_GSS_CREATE process subject label assertion is combined with a file object label provided by the NFSv4.2 sec_label attribute.

Label encoding is specified to mirror the NFSv4.2 sec_label attribute described in Section 12.2.4 of [NFSv4.2]. The label format specifier (LFS) is an identifier used by the client to establish the syntactic format of the security label and the semantic meaning of its components. The policy identifier (PI) is an optional part of the definition of an LFS which allows for clients and server to identify specific security policies. The opaque label field of rgss3_label is dependent on the MAC model to interpret and enforce.

If a label itself requires privacy protection (i.e., that the user can assert that label is a secret) then the client MUST use the rpc_gss_svc_privacy protection service for the RPCSEC_GSS_CREATE request.

RPCSEC_GSSv3 clients MAY assert a server security label in some LSF by binding a label assertion to the RPCSEC_GSSv3 context handle. This is done by including an assertion of type rgss3_label in the RPCSEC_GSS_CREATE rgss3_create_args rca_assertions call data.

Servers that support labeling in the requested LFS MAY map the requested subject label to a different subject label as a result of server-side policy evaluation.

The labels that are accepted by the target and bound to the RPCSEC_GSSv3 context MUST be enumerated in the rcr_assertions field of the rgss3_create_res RPCSEC_GSS_CREATE reply.

Servers that do not support labeling or that do not support the requested LFS reject the label assertion with a reply status of

MSG_DENIED, a reject_status of AUTH_ERROR, and an auth_stat of RPCSEC_GSS_LABEL_PROBLEM.

2.7.1.4. Structured Privilege Assertions

```
<CODE BEGINS>

   ///
   ///  struct rgss3_privs {
   ///          string          rp_name<>; /* human readable */
   ///          opaque          rp_privilege<>;
   ///  };

<CODE ENDS>
```

A structured privilege is an RPC application defined privilege. RPCSEC_GSSv3 clients MAY assert a structured privilege by binding the privilege to the RPCSEC_GSSv3 context handle. This is done by including an assertion of type rgss3_privs in the RPCSEC_GSS_CREATE rgss3_create_args rca_assertions call data. Encoding, server verification and any policies for structured privileges are described by the RPC application definition.

A successful structured privilege assertion MUST be enumerated in the rcr_assertions field of the rgss3_create_res RPCSEC_GSS_CREATE reply.

If a server receives a structured privilege assertion that it does not recognize the assertion is rejected with a reply status of MSG_DENIED, a reject_status of AUTH_ERROR, and an auth_stat of RPCSEC_GSS_UNKNOWN_MESSAGE.

If a server receives a structured privilege assertion that it fails to verify according to the requirements of the RPC application defined behavior, the assertion is rejected with a reply status of MSG_DENIED, a reject_status of AUTH_ERROR, and an auth_stat of RPCSEC_GSS_PRIVILEGE_PROBLEM.

Section 4.10.1.1. "Inter-Server Copy via ONC RPC with RPCSEC_GSSv3" of [NFSv4.2] shows an example of structured privilege definition and use.

2.7.2. New Control Procedure - RPCSEC_GSS_LIST

```
<CODE BEGINS>
```

```
   ///  enum rgss3_list_item {
   ///          LABEL = 0,
   ///          PRIVS = 1
   ///  };
   ///
   ///  struct rgss3_list_args {
   ///          rgss3_list_item      rla_list_what<>;
   ///  };
   ///
   ///  union rgss3_list_item_u
   ///        switch (rgss3_list_item itype) {
   ///  case LABEL:
   ///          rgss3_label          rli_labels<>;
   ///  case PRIVS:
   ///          rgss3_privs          rli_privs<>;
   ///  };
   ///
   ///  typedef rgss3_list_item_u rgss3_list_res<>;
   ///
```

<CODE ENDS>

The call data for an RPCSEC_GSS_LIST request consists of a list of integers (rla_list_what) indicating what assertions are to be listed, and the reply consists of an error or the requested list.

The result of requesting a list of rgss3_list_item LABEL is a list of LFSs supported by the server. The client can then use the LFS list to assert labels via the RPCSEC_GSS_CREATE label assertions. See Section 2.7.1.3.

2.8. Extensibility

Assertion types may be added in the future by adding arms to the 'rgss3_assertion_u' union. Examples of other potential assertion types include:

- Client-side assertions of identity:
  - Primary client/user identity
  - Supplementary group memberships of the client/user, including support for specifying deltas to the membership list as seen on the server.

3. Operational Recommendation for Deployment

RPCSEC_GSSv3 is a superset of RPCSEC_GSSv2 [RFC5403] which in turn is a superset of RPCSEC_GSSv1 [RFC2203], and so can be used in all situations where RPCSEC_GSSv1 or RPCSEC_GSSv2 is used. RPCSEC_GSSv3 should be used when the new functionality is needed.

4. Security Considerations

This entire document deals with security issues.

The RPCSEC_GSSv3 protocol allows for client-side assertions of data that is relevant to server-side authorization decisions. These assertions must be evaluated by the server in the context of whether the client and/or user are authenticated, whether multi-principal authentication was used, whether the client is trusted, what ranges of assertions are allowed for the client and the user (separately or together), and any relevant server-side policy.

The security semantics of assertions carried by RPCSEC_GSSv3 are application protocol-specific.

Note that RPSEC_GSSv3 is not a complete solution for labeling: it conveys the labels of actors, but not the labels of objects. RPC application protocols may require extending in order to carry object label information.

There may be interactions with NFSv4's callback security scheme and NFSv4.1's [RFC5661] GSS-API "SSV" mechanisms. Specifically, the NFSv4 callback scheme requires that the server initiate GSS-API security contexts, which does not work well in practice, and in the context of client- side processes running as the same user but with different privileges and security labels the NFSv4 callback security scheme seems particularly unlikely to work well. NFSv4.1 has the server use an existing, client-initiated RPCSEC_GSS context handle to protect server-initiated callback RPCs. The NFSv4.1 callback security scheme lacks all the problems of the NFSv4 scheme, however, it is important that the server pick an appropriate RPCSEC_GSS context handle to protect any callbacks. Specifically, it is important that the server use RPCSEC_GSS context handles which authenticate the client to protect any callbacks relating to server state initiated by RPCs protected by RPCSEC_GSSv3 contexts.

As described in Section 2.10.10 [RFC5661] the client is permitted to associate multiple RPCSEC_GSS handles with a single SSV GSS context. RPCSEC_GSSv3 handles will work well with SSV in that the man-in-the-middle attacks described in Section 2.10.10 [RFC5661] are solved by the new reply verifier (Section 2.3). Using an RPCSEC_GSSv3 handle

backed by a GSS-SSV mechanism context as a parent handle in an RPCSEC_GSS_CREATE call while permitted is complicated by the lifetime rules of SSV contexts and their associated RPCSEC_GSS handles.

5. IANA Considerations

IANA request #884160 is being processed for the new RPC authenticaion status numbers in Section 2.6.

6. References

6.1. Normative References

[NFSv4.2] Haynes, T., "NFS Version 4 Minor Version 2", draft-ietf-nfsv4-minorversion2-29 (Work In Progress), December 2014.

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

[RFC2203] Eisler, M., Chiu, A., and L. Ling, "RPCSEC_GSS Protocol Specification", RFC 2203, September 1997.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[RFC4506] Eisler, M., "XDR: External Data Representation Standard", RFC 4506, May 2006.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.

[RFC5403] Eisler, M., "RPCSEC_GSS Version 2", RFC 5403, February 2009.

[RFC5661] Shepler, S., Eisler, M., and D. Noveck, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, January 2010.

6.2. Informative References

[AFS-RXGK] Wilkinson, S. and B. Kaduk, "Integrating rxgk with AFS", draft-wilkinson-afs3-rxgk-afs (work in progress), April 2014.

[BL73] Bell, D. and L. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model", Technical Report M74-244, The MITRE Corporation Bedford, MA, May 1973.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[RFC4949] Shirley, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.

[RFC7204] Haynes, T., "Requirements for Labeled NFS", RFC 7204, April 2014.

Appendix A. Acknowledgments

Andy Adamson would like to thank NetApp, Inc. for its funding of his time on this project.

We thank Lars Eggert, Mike Eisler, Ben Kaduk, Bruce Fields, Tom Haynes, and Dave Noveck for their most helpful reviews.

Appendix B. RFC Editor Notes

[RFC Editor: please remove this section prior to publishing this document as an RFC]

[RFC Editor: prior to publishing this document as an RFC, please replace all occurrences of RFCTBD with RFCxxxx where xxxx is the RFC number of this document]

Authors' Addresses

William A. (Andy) Adamson
NetApp
3629 Wagner Ridge Ct
Ann Arbor, MI 48103
USA

Phone: +1 734 665 1204
Email: andros@netapp.com

Nico Williams
cryptonector.com
13115 Tamayo Dr
Austin, TX 78729
USA

Email: nico@cryptonector.com