NFSv4
Internet-Draft
Intended status: Standards Track
Expires: August 7, 2014

W. Adamson
NetApp
N. Williams
Cryptonector

February 03, 2014

Remote Procedure Call (RPC) Security Version 3
draft-ietf-nfsv4-rpcsec-gssv3-07.txt

Abstract

This document specifies version 3 of the Remote Procedure Call (RPC) security protocol (RPCSEC_GSS). This protocol provides for compound authentication of client hosts and users to 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 [1].

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 August 7, 2014.

Copyright Notice

Table of Contents

1. Introduction and Motivation

The original RPCSEC_GSS protocol [2] provided for authentication of RPC clients and servers to each other using the Generic Security Services Application Programming Interface (GSS-API) [3]. The second version of RPCSEC_GSS [4] added support for channel bindings [5].

We find that 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, but it seems be far simpler to address it at the application layer, namely, in this case, RPCSEC_GSS.

The motivation for RPCSEC_GSSv3 is to add support for labeled security and server-side copy for NFSv4.

Labeled NFS (see Section 8 of [6]) uses the subject label provided by the client via the RPCSEC_GSSv3 layer to enforce MAC access to objects owned by the server to enable server guest mode or full mode labeled NFS.

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 3.4.1 of [6]), the user first secures access to both source and destination files, and then uses RPCSEC_GSSv3 compound authentication and structured privileges to authorize the destination to copy the file from the source on behalf of the user.

We therefore describe RPCSEC_GSS version 3 (RPCSEC_GSSv3). RPCSEC_GSSv3 is the same as RPCSEC_GSSv2 [4], except that the following assertions of authority have been added.

- Security labels for multi-level, type enforcement, and other labeled security models. See [9], [10], [11], [6] and [12].

- Application-specific structured privileges. For an example see server-side copy [6].

- Compound authentication of the client host and user to the server done by binding two RPCSEC_GSS handles. For an example see server-side copy [6].

- 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.

We add an option for enumerating server supported label format specifiers (LFS). The LFS and Label Format Registry are described in detail in [13].

This document contains the External Data Representation (XDR) ([7]) 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_GSSv3 is the same as RPCSEC_GSSv2 [4], except that support for assertions has been added. The entire RPCSEC_GSSv3 protocol is not presented. Instead the differences between RPCSEC_GSSv3 and RPCSEC_GSSv2 are shown.

RPCSEC_GSSv3 is patterned as follows:

- A client uses an existing RPCSEC_GSSv3 context handle to protect RPCSEC_GSSv3 exchanges, this will be termed the "parent" handle. [[Comment.1: CAN A CHILD handle be used as a parent? --AA]]

- 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.

2.1. Compatibility with RPCSEC_GSSv2

The functionality of RPCSEC_GSSv2 [4] is fully supported by RPCSEC_GSSv3.

2.2. New REPLY verifier

The RPCSEC_GSSv3 child handle uses the same GSS context as the parent handle. Since a child and parent RPCSEC_GSSv3 handle could have the same RPCSEC_GSS sequence numbers, and the verifier of RPCSEC_GSS replies computes a MIC on just the sequence number, this provides opportunities for man in the middle attacks.

This is easily addressed: RPCSEC_GSS version 3 MUST change the verifier of the reply to compute the verifier using the exact same input as that is used for verifier of the request, except for the mtype change from CALL to REPLY:

```
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 */
```

2.3. New Version Number

<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 [RFCxxxx].
///   *
///   * 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]. 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 [2] and RPCSEC_GSSv2 [4] credential. Setting the rgc_version field to 3 indicates that the initiator and target support the new RPCSEC_GSSv3 control procedures.

2.4. 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: compound 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 version2 RPCSEC_GSS_DESTROY control message (see section 5.4 [2]) 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_channel_prot (see RPCSEC_GSSv2 [4])
- rpc_gss_svc_integrity
- rpc_gss_svc_privacy

Specifically the client MUST NOT use rpc_gss_svc_none.

2.4.1. New Control Procedure - RPCSEC_GSS_CREATE

<CODE BEGINS>

```
///  struct rgss3_create_args {
///          rgss3_gss_binding    *rca_comp_auth;
///          rgss3_chan_binding   *rca_chan_bind_mic;
///          rgss3_assertion      rca_assertions<>;
///  };
///
///  struct rgss3_create_res {
///          opaque               rcr_handle<>;
///          rgss3_gss_binding    *rcr_comp_auth;
///          rgss3_chan_binding   *rcr_chan_bind_mic;
///          rgss3_assertion      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<>;
///  };
///
///  struct rgss3_assertion {
///          bool                 ra_critical;
///          rgss3_assertion_u    ra_assertion;
///  };
///
```

<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:

- Compound authentication: another RPCSEC_GSS context handle
- Authorization assertions: labels and or privileges
- A channel binding

The reply to this message consists of either an error or an rgss3_create_res structure.

Upon successful RPCSEC_GSS_CREATE, both the client and the server SHOULD 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.

2.4.1.1. Compound Authentication

<CODE BEGINS>

```
///
///  struct rgss3_gss_binding {
///          opaque          rgb_handle<>; /* inner handle */
///          opaque          rgb_nonce<>;
///          opaque          rgb_nounc_mic<>;
///  };
///
```

<CODE ENDS>

RPCSEC_GSSv3 clients MAY assert a compound authentication of the client host and a user. This is done by including an assertion of type rgss3_gss_binding in the RPCSEC_GSS_CREATE rgss3_create_args call data. In addition to the parent handle (Section 2), the compound authentication rgss3_gss_binding call data has an RPCSEC_GSS

version 3 handle referenced via the rgb_handle field termed the "inner" handle. A nonce and a MIC of that nounce created using the GSS-API security context associated with the inner handle is also provided.

The target verifies the compounding by verifying the rgb_nouce_mic. On a successful reply, the rgss3_gss_binding field in the rgss3_create_res reply uses the parent RPCSEC_GSSv3 context as the rgb_handle, the same rgb_nounce as was sent in the call data with the rgb_nounce_mic created using the GSS-API security context associate with the parent handle. Verification of the rbg_nounce_mic by the initiator demonstrates that the target agrees to the compounding. On failure, the rgss3_gss_binding field is not sent. (rgss3_gss_binding is an optional field)

This feature is needed, for example, when a client wishes to use authority assertions that the server may only grant if a user and a client 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 a client acting alone (e.g. when a client is acting on behalf of a user) but may grant requested authority to a client acting on behalf of a user if the server identifies the user and trusts the client.

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

Clients using RPCSEC_GSSv3 compound authentication MUST use an RPCSEC_GSSv3 context handle that corresponds to a GSS-API security context that authenticates the client host for the outer handle. The inner context handle it SHOULD use a context handle to authenticate a user. The reverse (outer handle authenticates user, inner authenticates client) MUST NOT be used. Other compounds might eventually make sense.

An inner RPCSEC_GSSv3 context handle that is bound to an outer RPCSEC_GSS context 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.

2.4.1.2. 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 labels the server supports via the RPCSEC_GSS_LIST control message.

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.

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

Label encoding is specified to mirror the NFSv4.2 sec_label attribute described in Section 12.2.2 of [6]. 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.

Asserting a server supported label via RPCSEC_GSS_CREATE enables server guest mode labels. Full mode is enabled when an RPCSEC_GSS_CREATE label assertion is combined with asserting the same label with the NFSv4.2 sec_label attribute.

[[Comment.2: Check that this Label discussion provides all the required pieces to enable full mode when combined with NFSv4.2 LNFS. Specifically, how does the client find out and respond if a server has changed a label. --AA]]

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 or, if the parent handle is bound to a secure channel that provides privacy protection, rpc_gss_svc_channel_prot.

If a client wants to ensure that the server understands the asserted label then it MUST set the 'critical' field of the label assertion to TRUE, otherwise it MUST set it to FALSE.

Servers that do not support labeling MUST ignore non-critical label assertions. Servers that do not support the requested LFS MUST either ignore non-critical label assertions or map them to a suitable label in a supported LFS. Servers that do not support labeling or do not support the requested LFS MUST return an error if the label request is critical. Servers that support labeling in the requested LFS MAY map the requested label to different label as a result of server-side policy evaluation.

2.4.1.3. 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 will be enumerated in the rcr_assertions field of the rgss3_create_res RPCSEC_GSS_CREATE reply.

Section 3.4.1.2. "Inter-Server Copy with RPCSEC_GSSv3" of [6] shows an example of structured privilege definition and use.

2.4.1.4. Channel Binding

```
<CODE BEGINS>

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

<CODE ENDS>
```

RPCSEC_GSSv3 provides a different way to do channel binding than RPCSEC_GSSv2 [4]. 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 the community now agrees 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 [5], then the client calls GSS_GetMIC() to get a MIC of resulting octet string, using the 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, otherwise (if the client really wanted channel binding) it MUST call RPCSEC_GSS_DESTROY.

As per-RPCSEC_GSSv2 [4]:

> "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 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.

2.4.2. 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<>;
///  default:
///          opaque               rli_ext<>;
///  };
///
///  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 to be listed, and the reply consists of an error or the requested list.

[[Comment.3: What good is the rli_ext field? How should we describe it's use? --AA]]

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 2.4.1.2.

2.5. Extensibility

Assertion types may be added in the future by adding arms to the 'rgss3_assertion_u' union. Every assertion has a 'critical' flag that can be used to indicate criticality. Other assertion types are described elsewhere and 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.

New control message types may be added.

Servers receiving unknown critical client assertions MUST return an error.

2.6. New auth_stat Values

RPCSEC_GSSv3 requires the addition of several values to the auth_stat enumerated type definition:

```
enum auth_stat {
        ...
        /*
         * RPCSEC_GSSv3 errors
         */
        RPCSEC_GSS_COMPOUND_PROBLEM = <>,
        RPCSEC_GSS_LABEL_PROBLEM = <>,
        RPCSEC_GSS_UNKNOWN_PRIVILEGE = <>
        RPCSEC_GSS_UNKNOWN_MESSAGE = <>
```

```
        };
```

[[Comment.4: fix above into YYY. All the entries are TBD... --NW]]
[[Comment.5: The compound authentication problems are: can't find the handle plus handle version on the target, or the MIC of the nounce does not match. Both of these errors already have auth_stat entries: RPCSEC_GSS_CREDPROBLEM for the first and "reply status of MSG_ACCEPTED, and an acceptance status of GARBAGE_ARGS." --AA]]

3. 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 [2].

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.

4. Operational Recommendation for Deployment

RPCSEC_GSSv3 is a superset of RPCSEC_GSSv2 [4] which in turn is a superset of RPCSEC_GSSv1 [2], 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.

5. 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 compound 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.

RPCSEC_GSSv3 supports a notion of critical assertions but there's no need for peers to tell each other what assertions were granted, or what they were mapped to.

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 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.

[[Comment.6: [Add text about interaction with GSS-SSV...] --NW]]

[[Comment.7: AFAICS the reason to use SSV is to avoid using a client machine credential which means compound authentication can not be used. --AA]]

6. IANA Considerations

This section uses terms that are defined in [8].

There are no IANA considerations in this document. TBDs in this document will be assigned by the ONC RPC registrar (which is not IANA, XXX: verify).

7. References

7.1. Normative References

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

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

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

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

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

[6] Haynes, T., "NFS Version 4 Minor Version 2", draft-ietf-nfsv4-minorversion2-21 (Work In Progress), March 2013.

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

[8] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

7.2. Informative References

[9] "Section 46.6. Multi-Level Security (MLS) of Deployment Guide: Deployment, configuration and administration of Red Hat Enterprise Linux 5, Edition 6", 2011.

[10] Smalley, S., "The Distributed Trusted Operating System (DTOS) Home Page", <http://www.cs.utah.edu/flux/fluke/html/dtos/HTML/dtos.html>.

[11] Carter, J., "Implementing SELinux Support for NFS", <http://www.nsa.gov/research/_files/selinux/papers/nfsv3.pdf>.

[12] Haynes, T., "Requirements for Labeled NFS", draft-ietf-nfsv4-labreqs-05 (work in progress).

[13] Quigley, D. and J. Lu, "Registry Specification for MAC Security Label Formats", draft-quigley-label-format-registry (work in progress), 2011.

Appendix A. Acknowledgments

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 RFCTBD10 with RFCxxxx where xxxx is the RFC number of this document]

Authors' Addresses

William A. (Andy) Adamson
NetApp
3629 Wagner Ridge Ctt
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