Groupe de travail Réseau
Request for Comments : 2203
Catégorie : En cours de normalisation
Traduction Claude Brière de L'Isle

M. Eisler
A. Chiu
L. Ling
septembre 1997

Spécification du protocole RPCSEC_GSS

Résumé
Le présent mémoire décrit une nuance de sécurité ONC/RPC qui permet aux protocoles RPC d'accéder à l'interface de programmation d'application de services génériques de sécurité (GSS-API, *Generic Security Services Application Programming Interface*).

Table des Matières

1. Introduction

Le présent document décrit le protocole utilisé par la nuance de sécurité RPCSEC_GSS. Les nuances de sécurité ont été appelées nuances d'authentification pour des raisons historiques. Le présent mémoire reconnaît qu'il y a deux autres services de sécurité à côté de l'authentification, la protection de l'intégrité, la protection de la confidentialité, et il définit donc une nouvelle nuance de sécurité RPCSEC_GSS.

Le protocole est décrit en utilisant le langage XDR de la [RFC1832]. Le lecteur est supposé familier de ONC RPC et du mécanisme de nuance de sécurité de la [RFC1831]. Il est également supposé familier avec le cadre de GSS-API de la [RFC2078]. La nuance de sécurité RPCSEC_GSS utilise les interfaces GSS-API pour fournir des services de sécurité qui sont indépendants du mécanisme de sécurité sous-jacent.

2. Protocole de message ONC RPC

Le présent mémoire se réfère aux types XDR suivants du protocole ONC RPC, qui sont décrits dans le document intitulé "Spécification de la version 2 du protocole d'appel de procédure à distance" [RFC1831] :

```
msg_type
```

reply_stat
auth_flavor
accept_stat
reject_stat
auth_stat
opaque_auth
rpc_msg
call_body
reply_body
accepted_reply
rejected_reply

3. Allocation des numéros de nuance

La valeur 6 a été allouée à la nuance de sécurité RPCSEC_GSS :

```
enum auth_flavor {
   ...
   RPCSEC_GSS = 6                    /* nuance de sécurité RPCSEC_GSS */
};
```

4. Nouvelles valeurs de auth_stat

RPCSEC_GSS exige l'ajout de deux nouvelles valeurs à la définition des types auth_stat *(état d'authentification)* énumérés :

```
enum auth_stat {
    ...                                          /* erreurs RPCSEC_GSS */
    RPCSEC_GSS_CREDPROBLEM      = 13,
    RPCSEC_GSS_CTXPROBLEM       = 14
};
```

La description de ces deux nouvelles valeurs est donnée plus loin.

5. Éléments du protocole de sécurité RPCSEC_GSS

Une session RPC fondée sur la nuance de sécurité RPCSEC_GSS consiste en trois phases : création de contexte, échange de données RPC, et destruction de contexte. Dans l'exposé qui suit sont décrits les éléments de protocole pour ces trois phases.

La description suivante du protocole RPCSEC_GSS utilise certaines des définitions de la description en langage XDR du protocole RPC.

La création et la destruction de contexte utilisent des messages de contrôle qui ne sont pas envoyés aux procédures de service enregistrées par un serveur RPC. Le programme et les numéros de version utilisés dans ces messages de contrôle sont les mêmes que le programme et les numéros de version du service RPC. Le numéro de procédure utilisé est NULLPROC (zéro). Un champ dans les informations d'accréditifs (le champ gss_proc qui est défini ci-dessous dans la structure rpc_gss_cred_t) spécifie si un message doit être interprété comme message de contrôle ou comme message RPC régulier. Si ce champ est réglé à RPCSEC_GSS_DATA, aucune action de contrôle n'est impliquée ; dans ce cas, c'est un message de données régulier. Si ce champ est réglé à toute autre valeur, une action de contrôle est impliquée. Ceci est décrit dans les sections suivantes.

Tout comme avec les messages RPC normaux d'échange de données, l'identifiant de transaction (le champ xid dans la structure rpc_msg) devrait être réglé à des valeurs univoques sur chaque invocation de création de contexte et de destruction de contexte.

Les définitions suivantes sont utilisées pour décrire le protocole.

```
/* Procédures de contrôle de RPCSEC_GSS */

enum rpc_gss_proc_t {
      RPCSEC_GSS_DATA                = 0,
      RPCSEC_GSS_INIT                = 1,
      RPCSEC_GSS_CONTINUE_INIT       = 2,
      RPCSEC_GSS_DESTROY             = 3
};

/* services RPCSEC_GSS */

enum rpc_gss_service_t {                        /* Note : la valeur donnée pour 0 est réservée. */
   rpc_gss_svc_none                  = 1,
   rpc_gss_svc_integrity             = 2,
   rpc_gss_svc_privacy               = 3
};

/* Accréditifs */

        /* Note : la version 0 est réservée pour de possibles futures définitions d'un protocole de négociation de version */

#define RPCSEC_GSS_VERS_1 1

struct rpc_gss_cred_t {
  union switch (unsigned int version) {          /* version de RPCSEC_GSS */
    case RPCSEC_GSS_VERS_1:
      struct {
        rpc_gss_proc_t gss_proc;                 /* procédure de contrôle */
        unsigned int seq_num;                    /* numéro de séquence */
        rpc_gss_service_t service;               /* service utilisé */
        opaque handle<>;                         /* lien de contexte */
        } rpc_gss_cred_vers_1_t;
    }
};

/* Valeur maximum de numéro de séquence */

#define MAXSEQ 0x80000000
```

5.1 Choix de version

Le présent document ne définit qu'une seule version de protocole (RPCSEC_GSS_VERS_1). Le client devrait supposer que le serveur prend en charge RPCSEC_GSS_VERS_1 et produit un message Création de contexte (comme décrit dans la section RPCSEC_GSS_VERS_1, la réponse RPC aura un reply_stat *(état de réponse)* de MSG_DENIED, un reject_stat *(état de rejet)* de AUTH_ERROR, et un auth_stat *(état d'authentification)* de AUTH_REJECTED_CRED.

5.2 Création de contexte

Avant que les données RPC soient échangées dans une session qui utilisé la nuance RPCSEC_GSS, un contexte doit être établi entre le client et le serveur. la création de contexte peut impliquer zéro, un ou plusieurs échanges RPC. Le nombre d'échanges dépend du mécanisme de sécurité.

5.2.1 Mécanisme et choix de QOP

Il n'y a pas de facilité dans le protocole RPCSEC_GSS pour négocier des identifiants de mécanisme GSS-API ou de valeurs de qualité de protection (QOP, *quality of protection*). Au minimum, on attend des mises en œuvre du protocole RPCSEC_GSS qu'elles fournissent les moyens :

* de spécifier des identifiants de mécanisme, des valeurs de QOP, et des valeurs de service RPCSEC_GSS sur le côté client, et
* de mettre en application des identifiants de mécanisme, des valeurs de QOP, et des valeurs de service RPCSEC_GSS à

la demande du côté serveur.

Il est nécessaire que les capacités ci-dessus existent afin que les applications aient les moyens de se conformer à l'ensemble requis de triplets de <mécanisme, QOP, service> (voir la section intitulée "Ensemble de mécanismes GSS-API"). Une application peut négocier un choix de <mécanisme, QOP, service> au sein de son protocole ou via un protocole hors bande. Donc, il peut être nécessaire que les mises en œuvre RPCSEC_GSS fournissent les interfaces de programmation pour la spécification et la mise en application du triplet <mécanisme, QOP, service>.

De plus, les mises en œuvre peuvent dépendre des schémas de négociation construits comme des pseudo-mécanismes sous la GSS-API. Parce que de tels schémas sont en dessous de la couche GSS-API, le protocole RPCSEC_GSS, comme spécifié dans le présent document, peut en faire usage.

5.2.2 Demandes de création de contexte

La première demande RPC du client au serveur initie la création de contexte. Au sein de la structure call_body *(corps d'appel)* du protocole de message RPC, rpcvers est réglé à 2. prog et vers sont toujours ceux pour le service auquel on accède. Le champ proc est toujours réglé à NULLPROC (zéro).

Au sein de la structure cred *(accréditifs)* du protocole de message RPC, la nuance est réglée à RPCSEC_GSS (6). Les données opaques de la structure cred (le champ corps) constituant l'accréditif codent la structure rpc_gss_cred_t définie précédemment.

Les valeurs des champs contenus dans la structure rpc_gss_cred_t sont toujours réglées comme suit. Le champ version est réglé à la version du protocole RPCSEC_GSS que le client veut utiliser. Le reste du présent mémoire documente la version RPCSEC_GSS_VERS_1 de RPCSEC_GSS, et donc le champ version sera réglé à RPCSEC_GSS_VERS_1. Le champ gss_proc doit être réglé à RPCSEC_GSS_INIT pour la première demande de création. Dans les demandes de création suivantes, le champ gss_proc doit être réglé à RPCSEC_GSS_CONTINUE_INIT. Dans une demande de création, les champs seq_num et service sont indéfinis et tous deux doivent être ignorés par le serveur. Dans la première demande de création, le champ lien *(handle)* est NUL (données opaques de longueur zéro). Dans les demandes de création suivantes, handle doit être égal à la valeur retournée par le serveur. Le champ handle sert d'identifiant pour le contexte, et ne va pas changer pour la durée du contexte, y compris pour les réponses à RPCSEC_GSS_CONTINUE_INIT.

Le champ verifier *(vérificateur)* dans l'en-tête de message RPC est aussi décrit par la structure opaque_auth. Toutes les demandes de création ont le vérificateur NUL (nuance AUTH_NONE avec les données opaques de longueur zéro).

Après le vérificateur se trouvent les données d'appel (paramètres spécifiques de la procédure). Noter que le champ proc de la structure call_body est réglé à NULLPROC, que donc normalement il y aura zéro octet à la suite du vérificateur. Cependant, comme il n'y a pas d'échange de données RPC durant une création de contexte, il est sûr de transférer les informations qui suivent le vérificateur. Il est nécessaire de "surcharger" les données d'appel de cette façon, plutôt que d'empaqueter le jeton GSS-API dans l'en-tête RPC, parce que RPC version 2 restreint la quantité de données qui peuvent être envoyées dans l'en-tête. Le corps opaque des champs accréditif et vérificateur peut être au plus de 400 octets chacun, et les jetons GSS peuvent faire plus de 800 octets.

L'appel de données pour une demande de création de contexte est décrit par la structure suivante pour toute demande de création :

```
struct rpc_gss_init_arg {
   opaque gss_token<>;
};
```

Ici, gss_token est le jeton retourné par l'invocation du sous-programe GSS_Init_sec_context() de GSS-API, codé de façon opaque. La valeur de ce champ sera probablement différente dans chaque demande de création, si il y a plus d'une demande de création. Si aucun jeton n'est retourné par l'invocation de GSS_Init_sec_context(), le contexte doit avoir été créé (en supposant qu'il n'y a pas d'erreur) et il n'y aura plus d'autre demande de création.

Lorsque GSS_Init_sec_context() est invoqué, les paramètres replay_det_req_flag et sequence_req_flag doivent être mis à zéro. La raison en est que :

* ONC RPC peut être utilisé sur des transports non fiables et ne fournit pas de couche pour réassembler fiablement les messages. Donc il est possible que surviennent des trous dans la séquence des messages, ainsi que des messages déclassés.
* Les serveurs RPC peuvent être multi-trames, et donc l'ordre dans lequel les messages GSS-API sont signés ou enveloppés peut être différent de l'ordre dans lequel les messages sont vérifiés ou désenveloppés, même si les

demandes sont envoyées sur des transports fiables.

* Pour maximiser la facilité de mise en œuvre, l'ordre dans lequel une entité ONC RPC va vérifier l'en-tête et vérifier/désenvelopper le corps d'un appel ou réponse RPC n'est pas spécifié.

Le protocole RPCSEC_GSS assure la protection contre les attaques en répétition, et donc tolère la livraison ou le traitement de messages déclassés et tolère les abandons de demandes.

5.2.3 Réponses de création de contexte

5.2.3.1 Réponses de création de contexte – acceptation réussie

La réponse à une demande de création réussie a une réponse MSG_ACCEPTED avec un état de SUCCESS. Le champ results code une réponse avec la structure suivante :

```
struct rpc_gss_init_res {
     opaque handle<>;
     unsigned int gss_major;
     unsigned int gss_minor;
     unsigned int seq_window;
     opaque gss_token<>;
};
```

Ici, handle sont des données opaques non NULLES qui servent d'identifiant de contexte. Le client doit utiliser cette valeur dans toutes les demandes suivantes qu'elles soient ou non des messages de contrôle). Les champs gss_major et gss_minor contiennent le résultat des invocations de GSS_Accept_sec_context() exécutées par le serveur. Les valeurs pour le champ gss_major sont définies dans l'Appendice A du présent document. Les valeurs pour le champ gss_minor sont spécifiques du mécanisme GSS-API et sont définies dans la spécification du mécanisme. Si gss_major n'est pas GSS_S_COMPLETE ou GSS_S_CONTINUE_NEEDED, l'établissement de contexte a échoué ; dans ce cas, handle et gss_token doivent être réglés à NUL par le serveur. La valeur de gss_minor dépend de la valeur de gss_major du mécanisme de sécurité utilisé. Le champ gss_token contient tout jeton retourné par l'invocation de GSS_Accept_sec_context() exécutée par le serveur. Un jeton peut être retourné pour les deux valeurs de réussite de gss_major. Si la valeur est GSS_S_COMPLETE, cela indique que le serveur n'attend plus d'autre jeton, et que la phase d'échange de données RPC doit commencer sur la demande suivante du client. Si la valeur est GSS_S_CONTINUE_NEEDED, le serveur attend un autre jeton. Donc, le client doit envoyer au moins une autre demande de création (avec gss_proc réglé à RPCSEC_GSS_CONTINUE_INIT dans l'accréditif de la demande) portant le jeton requis.

Dans une réponse réussie, le champ seq_window est réglé à la longueur de fenêtre de séquence acceptée par le serveur pour ce contexte. Cette fenêtre spécifie le nombre maximum de demandes du client qui peuvent être en instance pour ce contexte. Le serveur va accepter "seq_window" demandes à la fois, et elles peuvent être dans le désordre. Le client peut utiliser ce nombre pour déterminer le nombre de trames qui peuvent envoyer simultanément des demandes sur ce contexte.

Si gss_major est GSS_S_COMPLETE, le champ nuance du vérificateur (l'élément verf dans la réponse) est réglé à RPCSEC_GSS, et le champ corps est réglé à la somme de contrôle de seq_window (dans l'ordre des octets du réseau). La QOP utilisée pour cette somme de contrôle est 0 (zéro), qui est la QOP par défaut. Pour toutes les autres valeurs de gss_major, un vérificateur NUL (nuance AUTH_NONE avec des données opaques de longueur zéro) est utilisé.

5.2.3.1.1 Traitement par le client des réponses de création de contexte réussies

Si la valeur de gss_major dans la réponse est GSS_S_CONTINUE_NEEDED, alors le client, selon la spécification GSS-API, doit invoquer GSS_Init_sec_context() en utilisant le jeton retourné dans gss_token dans la réponse de création de contexte. Le client doit alors générer une demande de création de contexte, avec gss_proc réglé à RPCSEC_GSS_CONTINUE_INIT.

Si la valeur de gss_major dans la réponse est GSS_S_COMPLETE, et si l'invocation précédente du client de GSS_Init_sec_context() a retourné une valeur de gss_major de GSS_S_CONTINUE_NEEDED, alors le client, selon la spécification GSS-API, doit invoquer GSS_Init_sec_context() en utilisant le jeton retourné dans gss_token dans la réponse de création de contexte. Si GSS_Init_sec_context() retourne GSS_S_COMPLETE, le contexte est bien établi, et la phase d'échange de données RPC doit commencer sur la demande suivante du client.

5.2.3.2 Réponse de création de contexte – cas d'échec

Une réponse MSG_ACCEPTED (à une demande de création) avec un état d'acceptation autre que SUCCESS a un vérificateur NUL (nuance réglée à AUTH_NONE, et des données opaques de longueur zéro dans le champ de corps) et est formulée comme d'habitude pour les valeurs d'état différentes.

Une réponse MSG_DENIED (à une demande de création) est aussi formulée comme d'habitude. Noter que MSG_DENIED pourrait être retourné parce que la mise en œuvre de RPC du serveur ne reconnaît pas la nuance de sécurité RPCSEC_GSS. La [RFC1831] ne spécifie pas l'état de réponse appropriée dans cette instance, mais la pratique courante des mises en œuvre parait être de retourner un état de rejet de AUTH_ERROR avec un auth_stat de AUTH_REJECTEDCRED. Bien que deux nouvelles valeurs (CSEC_GSS_CREDPROBLEM et RPCSEC_GSS_CTXPROBLEM) aient été définies pour le type auth_stat, aucune d'elles ne peut être retournée en réponse aux demandes de création de contexte. Les nouvelles valeurs de auth_stat peuvent être utilisées pour les réponses aux demandes normales (de données). Ceci sera décrit plus loin.

MSG_DENIED pourrait aussi être retourné si le numéro de version RPCSEC_GSS dans l'accréditif n'est pas accepté sur le serveur. Dans ce cas, le serveur retourne un état de rejet de AUTH_ERROR, avec un auth_stat de AUTH_REJECTED_CRED.

5.3 Échange de données RPC

On entre dans la phase d'échange de données après la réussite d'un établissement de contexte. Le format des données échangées dépend du service de sécurité utilisé pour la demande. Bien que les clients puissent changer le service de sécurité et la QOP utilisés demande par demande, cela peut n'être pas acceptable à tous les services RPC ; certains services RPC peuvent "verrouiller" la phase d'échange de données en utilisant la QOP et le service utilisé sur le premier message d'échange de données. Pour tous les trois modes de service (pas d'intégrité des données, intégrité des données, confidentialité des données) l'en-tête de demande RPC a le même format.

5.3.1 En-tête de demande RPC

L'accréditif a la structure opaque_auth décrite précédemment. Le champ nuance est réglé à RPCSEC_GSS. Le corps de l'accréditif est créé en codant en XDR la structure rpc_gss_cred_t mentionnée plus haut dans un flux d'octets, et ensuite en codant de façon opaque ce flux d'octets comme champ de corps.

Les valeurs des champs contenus dans la structure rpc_gss_cred_t sont réglés comme suit : le champ de version est réglé à la même valeur de version qu'utilisée pour créer le contexte, qui dans le domaine d'application du présent mémoire sera toujours RPCSEC_GSS_VERS_1. Le champ gss_proc est réglé à RPCSEC_GSS_DATA. Le champ service est réglé à indiquer le service désiré (rpc_gss_svc_none, rpc_gss_svc_integrity, ou rpc_gss_svc_privacy). Le champ handle est réglé à la valeur de lien de contexte reçue du serveur RPC durant la création du contexte. Le champ seq_num peut commencer à toute valeur en dessous de MAXSEQ, et doit être incrémentée (de un ou plus) pour les demandes successives. L'utilisation des numéros de séquence est décrite en détail lors de l'exposé sur le traitement de la demande.

Le vérificateur a la structure opaque_auth décrite précédemment. Le champ nuance est réglé à RPCSEC_GSS. Le champ corps est réglé comme suit. La somme de contrôle de l'en-tête RPC (jusque et y compris l'accréditif) est calculé en utilisant l'invocation de GSS_GetMIC() avec la QOP désirée. Cela retourne la somme de contrôle comme flux d'octets opaque et sa longueur. Ceci est codé dans le champ corps. Noter que la QOP n'est spécifiée explicitement nulle part dans la demande. Elle est implicite dans la somme de contrôle ou dans les données chiffrées. La même valeur de QOP qu'utilisée pour la somme de contrôle d'en-tête doit aussi être utilisée pour les données (pour la somme de contrôle ou pour le chiffrement) sauf si le service utilisé pour la demande est rpc_gss_svc_none.

5.3.2 Données de demande RPC

5.3.2.1 Données de demande RPC – pas d'intégrité des données

Si le service spécifié est rpc_gss_svc_none, les données (arguments de procédure) n'ont pas de protection d'intégrité ni de confidentialité. Elle sont envoyées exactement de la même façon qu'elles l'auraient été si la nuance AUTH_NONE avait été utilisée (suivant le vérificateur). Noter cependant que comme l'en-tête RPC est protégé en intégrité, l'envoyeur va quand même être authentifié dans ce cas.

5.3.2.2 Données de demande RPC – avec intégrité des données

Lorsque on utilise l'intégrité des données, les données de demande sont représentées comme suit :

```
struct rpc_gss_integ_data {
   opaque databody_integ<>;
   opaque checksum<>;
};
```

Le champ databody_integ est créé comme suit. On construit une structure consistant en un numéro de séquence suivi par les arguments de procédures. Ceci est montré ci-dessous comme le type rpc_gss_data_t :

```
struct rpc_gss_data_t {
   unsigned int seq_num;
   proc_req_arg_t arg;
};
```

Ici, seq_num doit avoir la même valeur que dans l'accréditif. Le type proc_req_arg_t est le type XDR spécifique de la procédure qui décrit les arguments de procédure (qui ne sont pas spécifiés ici). Le flux d'octets qui correspond à la structure codée en XDR rpc_gss_data_t et sa longueur est placé dans le champ databody_integ. Noter que comme le type XDR de databody_integ est opaque, le codage XDR de databody_integ va inclure un champ initial de quatre octets, suivi par le flux d'octets codé en XDR de rpc_gss_data_t.

Le champ somme de contrôle représente la somme de contrôle du flux d'octets codé en XDR correspondant à la structure rpc_gss_data_t codée en XDR (noter que ceci n'est pas la somme de contrôle du champ databody_integ). On obtient ceci en utilisant l'invocation de GSS_GetMIC(), avec la même QOP qui a été utilisée pour calculer la somme de contrôle de l'en-tête (dans le vérificateur). L'invocation de GSS_GetMIC() retourne la somme de contrôle comme un flux d'octets opaque et sa longueur. Le champ somme de contrôle de la structure rpc_gss_integ_data a un type XDR de opaque. Donc, la longueur de la somme de contrôle tirée de GSS_GetMIC() est codée comme un champ de quatre octets, suivi par la somme de contrôle, bourrée à un multiple de quatre octets.

5.3.2.3 Données de demande RPC – avec confidentialité des données

Lorsque on utilise la confidentialité des données, les données de la demande sont représentées comme suit :

```
struct rpc_gss_priv_data {
   opaque databody_priv<>
};
```

Le champ databody_priv est créé comme suit. La structure rpc_gss_data_t décrite précédemment est construite de la même façon que dans le cas de l'intégrité des données. Ensuite, l'invocation de GSS_Wrap() est faite pour chiffrer le flux d'octets correspondant à la structure rpc_gss_data_t, en utilisant la même valeur de QOP (l'argument qop_req de GSS_Wrap()) qui a été utilisé pour la somme de contrôle d'en-tête (dans le vérificateur) et de conf_req_flag (un argument de GSS_Wrap()) de VRAI. L'invocation de GSS_Wrap() retourne un flux d'octets opaque (représentant la structure rpc_gss_data_t chiffrée) et sa longueur, et cela est codé comme champ databody_priv. Comme databody_priv a un type XDR de opaque, la longueur retournée par GSS_Wrap() est codée comme longueur sur quatre octets, suivis par le flux d'octets chiffré (bourré jusqu'à un multiple de quatre octets).

5.3.3 Traitement des demandes de données RPC au serveur

5.3.3.1 Gestion de contexte

Lorsque une demande est reçue par le serveur, l'acceptabilité de ce qui suit est vérifiée :

* le numéro de version dans l'accréditif,
* le service spécifié dans l'accréditif,
* le lien de contexte spécifié dans l'accréditif,
* la somme de contrôle de l'en-tête dans le vérificateur (via GSS_VerifyMIC())
* le numéro de séquence (seq_num) spécifié dans l'accréditif (ce point est développé ensuite).

Le champ gss_proc dans l'accréditif doit être réglé à RPCSEC_GSS_DATA pour les demandes de données (autrement, le message sera interprété comme un message de contrôle).

Le serveur entretient une fenêtre de "seq_window" numéros de séquence, commençant par le dernier numéro de séquence vu et s'étendant en arrière. Si un numéro de séquence supérieur au dernier numéro vu est reçu (ET si GSS_VerifyMIC() sur la somme de contrôle d'en-tête venant du vérificateur retourne GSS_S_COMPLETE) la fenêtre est remontée au nouveau numéro de séquence. Si le dernier numéro de séquence vu est N, le serveur est prêt à recevoir des demandes avec des numéros de séquence dans la gamme de N à (N - seq_window + 1), tous deux inclus. Si le numéro de séquence reçu tombe en dessous de cette gamme, il est éliminé en silence. Si le numéro de séquence est dans cette gamme, et si le serveur ne l'a pas vu, la demande est acceptée, et le serveur change un bit pour "se rappeler" que ce numéro de séquence a été vu. Si le serveur détermine qu'il a déjà vu un numéro de séquence au sein de la fenêtre, la demande est éliminée en silence. Le serveur devrait choisir une valeur de seq_window sur la base du nombre de demandes qu'il s'attend à traiter simultanément. Par exemple, dans une mise en œuvre en trames, seq_window peut être égal au nombre de trames de serveur. Il n'y a pas de problèmes de sécurité connus pour le choix d'une grande fenêtre. La principale question est celle de la quantité d'espace que le serveur veut allouer pour garder trace des demandes reçues au sein de la fenêtre.

La raison de l'élimination en silence des demandes est que le serveur n'est pas capable de déterminer si la demande dupliquée ou déclassée est due à un problème de séquençage chez le client, le réseau, ou le système d'exploitation, ou due à un caprice de l'acheminement, ou une attaque en répétition lancée par un intrus. Éliminer la demande en silence permet au client de récupérer après une temporisation, si bien sûr la duplication était non intentionnelle ou bien volontaire. Noter qu'une conséquence de l'élimination en silence est que les clients peuvent incrémenter le seq_num de plus de un. L'effet en est que la fenêtre va avance plus vite. On ne sait pas quel avantage on peut en tirer.

Noter que l'algorithme de numéro de séquence exige que le client incrémente le numéro de séquence même si il réessaye une demande avec le même identifiant de transaction RPC. Il n'est pas rare que les clients se trouvent dans une situation où ils envoient deux tentatives ou plus et où un serveur lent envoie la réponse pour la première tentative. Avec RPCSEC_GSS, chaque demande et chaque réponse va avoir un numéro de séquence univoque. Si le client souhaite améliorer le temps de traitement des invocations de RPC, il peut mettre en antémémoire le numéro de séquence RPCSEC_GSS de chaque demande qu'il envoie. Puis quand il reçoit une réponse avec un identifiant de transaction RPC qui correspond, il peut calculer la somme de contrôle de chaque numéro de séquence qui est dans l'antémémoire pour essayer de faire correspondre la somme de contrôle dans le vérificateur de la réponse.

Les données sont décodées conformément au service spécifié dans l'accréditif. Dans le cas de l'intégrité ou de la confidentialité, le serveur s'assure que la valeur de QOP est acceptable, et qu'elle est la même que celle utilisée pour la somme de contrôle d'en-tête dans le vérificateur. Aussi, dans le cas de l'intégrité ou de la confidentialité, le serveur va rejeter le message (avec un état de réponse de MSG_ACCEPTED, et un état d'acceptation de GARBAGE_ARGS) si le numéro de séquence incorporé dans le corps de la demande est différent de celui de l'accréditif.

5.3.3.2 Réponse du serveur – demande acceptée

Une réponse MSG_ACCEPTED à une demande dans la phase d'échange des données aura le champ nuance du vérificateur (l'élément verf dans la réponse) réglé à RPCSEC_GSS, et le champ corps réglé à la somme de contrôle (le résultat de GSS_GetMIC()) du numéro de séquence (dans l'ordre des octets du réseau) de la demande correspondante. La QOP utilisée est la même que celle de la demande correspondante.

Si le statut de la réponse n'est pas SUCCESS, le reste du message est formaté comme d'habitude.

Si le statut du message est SUCCESS, le format du reste du message dépend du service spécifié dans le message de demande correspondant. Fondamentalement, ce qui suit le vérificateur dans ce cas sont les résultats de la procédure, formatés de différentes façons selon le service demandé.

Si la protection de l'intégrité des données n'était pas demandée, les résultats de la procédure sont formatés comme pour la nuance de sécurité AUTH_NONE.

Si l'intégrité des données était demandée, le résultat est codé exactement de la même façon que l'étaient les arguments de procédure dans la demande correspondante. Voir le paragraphe "Données de demande RPC - avec intégrité des données". La seule différence est que la structure représentant le résultat de la procédure - proc_res_arg_t - doit être substitué à la structure d'argument de demande proc_req_arg_t. La QOP utilisée pour la somme de contrôle doit être la même que celle utilisée pour construire le vérificateur de la réponse.

Si la confidentialité des données était demandée, les résultats sont codés exactement de la même façon que l'étaient les arguments de procédure dans la demande correspondante. Voir le paragraphe "Données de demande RPC – avec confidentialité des données". La QOP utilisée pour le chiffrement doit être la même que celle utilisée pour construire le vérificateur de réponse.

5.3.3.3 Réponse du serveur – demande refusée

Une réponse MSG_DENIED (à une demande de données) est formulée comme d'habitude. On a défini deux nouvelles valeurs (RPCSEC_GSS_CREDPROBLEM et RPCSEC_GSS_CTXPROBLEM) pour le type auth_stat. Lorsque la raison du refus de la demande est un reject_stat de AUTH_ERROR, une des deux nouvelles valeurs de auth_stat pourrait être retournée en plus des valeurs existantes. Ces deux nouvelles valeurs ont une signification particulière parmi les raisons existantes de refus d'une demande.

Le serveur entretient une liste des contextes pour les clients qui sont actuellement en session avec lui. Normalement, un contexte est détruit lorsque le client met un terme à la session qui lui correspond. Cependant, du fait des contraintes de ressources, le serveur peut détruire prématurément un contexte (sur la base du moins récemment utilisé, ou si la machine du serveur est réamorcée, par exemple). Dans ce cas, lorsque une demande de client arrive, il peut ne pas y avoir de contexte correspondant à son lien. Le serveur rejette la demande, avec la cause RPCSEC_GSS_CREDPROBLEM dans ce cas. À réception de cette erreur, le client doit rafraîchir le contexte – c'est-à-dire, le rétablir après avoir détruit le vieux - et essayer à nouveau la demande. Cette erreur est aussi retournée si le lien du contexte correspond à celui d'un contexte différent qui a été alloué après la destruction du contexte du client (cela sera détecté par un échec de la vérification de la

somme de contrôle d'en-tête).

Si l'invocation de GSS_VerifyMIC() sur la somme de contrôle d'en-tête (contenue dans le vérificateur) échoue à retourner GSS_S_COMPLETE, le serveur rejette la demande et retourne un auth_stat de RPCSEC_GSS_CREDPROBLEM.

Lorsque le numéro de séquence du client excède le maximum que permet le serveur, celui-ci va rejeter la demande avec la cause RPCSEC_GSS_CTXPROBLEM. Aussi, si les accréditifs de sécurité deviennent périmés pendant leur utilisation (due à l'arrivée à expiration du ticket dans le cas du mécanisme Kerberos V5, par exemple) les échecs qui en résultent causent le retour de la cause RPCSEC_GSS_CTXPROBLEM. Dans ces cas aussi, le client doit rafraîchir le contexte, et réessayer la demande.

Pour les autres erreurs, réessayer ne va pas rectifier le problème et le client ne doit pas rafraîchir le contexte tant que le problème qui cause le refus de la demande du client n'est pas corrigé.

Si le champ version dans l'accréditif ne correspond pas à la version de RPCSEC_GSS qui a été utilisée lors de la création du contexte, la valeur AUTH_BADCRED est retournée.

Si il y a un problème avec l'accréditif, comme une mauvaise longueur, une procédure de contrôle illégale, ou un service illégal, l'état auth_state approprié est AUTH_BADCRED.

D'autres erreurs peuvent être retournées comme approprié.

5.3.3.4 Transposition des erreurs GSS-API en réponses du serveur

Durant la phase d'échange des données, le serveur peut invoquer GSS_GetMIC(), GSS_VerifyMIC(), GSS_Unwrap(), et GSS_Wrap(). Si un de ces sous-programmes échoue à retourner GSS_S_COMPLETE, diverses réponses d'échec peuvent alors être retournées. Elles sont décrites comme suit pour chacune des quatre interfaces mentionnées ci-dessus.

5.3.3.4.1 Échec de GSS_GetMIC()

Lorsque GSS_GetMIC() est invoqué pour générer le vérificateur dans la réponse, un échec résulte en une réponse RPC avec un état de réponse de MSG_DENIED, un état de rejet de AUTH_ERROR et un état d'authentification de RPCSEC_GSS_CTXPROBLEM.

Lorsque GSS_GetMIC() est invoqué pour signer les résultats de l'invocation (le service est rpc_gss_svc_integrity) un échec ne résulte en l'envoi d'aucune réponse RPC. Comme les applications de serveur RPC ONC vont normalement contrôler quand une réponse est envoyée, l'indication d'échec sera retournée à l'application de serveur et elle peut prendre l'action appropriée (comme d'inscrire l'erreur dans le journal d'événements).

5.3.3.4.2 Échec de GSS_VerifyMIC()

Lorsque GSS_VerifyMIC() est invoqué pour vérifier le vérificateur dans la demande, un échec résulte en une réponse RPC avec un état de réponse de MSG_DENIED, un état de rejet de AUTH_ERROR et un état d'authentification de RPCSEC_GSS_CREDPROBLEM.

Lorsque GSS_VerifyMIC() est invoqué pour vérifier les arguments d'appel (le service est rpc_gss_svc_integrity) un échec résulte en une réponse RPC avec un état de réponse de MSG_ACCEPTED, et un état d'acceptation de GARBAGE_ARGS.

5.3.3.4.3 Échec de GSS_Unwrap()

Lorsque GSS_Unwrap() est invoqué pour déchiffrer les arguments d'appel (le service est rpc_gss_svc_privacy) un échec résulte en une réponse RPC avec un état de réponse de MSG_ACCEPTED, et un état d'acceptation de GARBAGE_ARGS.

5.3.3.4.4 Échec de GSS_Wrap()

Lorsque GSS_Wrap() est invoqué pour chiffrer les résultats de l'appel (le service est rpc_gss_svc_privacy) un échec ne résulte en l'envoi d'aucune réponse RPC. Comme les applications de serveur RPC ONC vont normalement contrôler quand une réponse est envoyée, l'indication d'échec sera retournée à l'application et elle peut prendre l'action appropriée (comme d'inscrire l'erreur dans le journal d'événements).

5.4 Destruction de contexte

Lorsque le client a fini d'utiliser la session, il doit envoyer un message de contrôle qui informe le serveur qu'il n'a plus besoin du contexte. Ce message est formulé tout comme un paquet de demande de données, avec les différences suivantes : l'accréditif a gss_proc réglé à RPCSEC_GSS_DESTROY, la procédure spécifiée dans l'en-tête est NULLPROC, et il n'y a pas d'argument de procédure. Le numéro de séquence dans la demande doit être valide, et la somme de contrôle de l'en-tête dans le vérificateur doit être valide, pour que le serveur accepte le message. Le serveur envoie une réponse comme il ferait

pour une demande de données. Le client et le serveur doivent alors détruire le contexte pour la session.

Si la demande de destruction du contexte échoue pour une raison quelconque, le client n'a pas besoin de prendre de mesure particulière. Le serveur doit être prêt à traiter des situations où les clients ne les informent jamais qu'ils ne sont plus en session et n'ont donc plus besoin que le serveur entretienne un contexte. Un mécanisme de LRU *(Least Recently Used, le moins récemment utilisé)* ou un mécanisme de péremption devrait être employé par le serveur pour faire le ménage dans de tels cas.

6. Ensemble de mécanismes de GSS-API

RPCSEC_GSS est effectivement une "passerelle" pour la couche GSS-API, et à ce titre il est inapproprié que la spécification de RPCSEC_GSS énumère un ensemble minimum de mécanismes exigés de sécurité et/ou de qualité de protection.

Si une spécification de protocole d'application fait référence à RPCSEC_GSS, la spécification du protocole doit faire la liste d'un ensemble obligatoire de triplets de {mécanisme, QOP, service} tel qu'une mise en œuvre ne puisse revendiquer la conformité à cette spécification de protocole si elle ne met pas en œuvre l'ensemble de triplets. Au sein de chaque triplet, le mécanisme est un mécanisme de sécurité GSS-API, la QOP est une qualité de protection valide au sein du mécanisme, et le service est soit rpc_gss_svc_integrity, soit rpc_gss_svc_privacy.

Par exemple, un protocole de passage en réseau construit sur RPC qui dépend de RPCSEC_GSS pour sa sécurité, peut exiger que Kerberos v5 avec la QOP par défaut utilisant le service rpc_gss_svc_integrity soit pris en charge par les mises en œuvre conformes à la spécification de protocole de passage en réseau.

7. Considérations pour la sécurité

7.1 Confidentialité de l'en-tête d'invocation

Le lecteur aura noté que pour l'option confidentialité, seuls les arguments et résultats de l'invocation sont chiffrés. Les informations sur l'application sous la forme de numéro de programme RPC, de numéro de version de programme, et de numéro de procédure de programme sont transmises en clair. Le chiffrement de ces champs dans l'en-tête d'invocation RPC changerait la taille et le format de l'en-tête de l'invocation. Cela aurait exigé de réviser le protocole RPC, ce qui sortait du domaine d'application de la présente proposition. Mémoriser les numéros chiffrés dans l'accréditif aurait évité un changement du protocole, mais aurait introduit plus de surcharge des champs et aurait rendues plus complexes les mises en œuvre de RPC. Même si les champs étaient chiffrés, dans la plupart des cas, un attaquant pourrait déterminer le numéro de programme et le numéro de version en examinant l'adresse de destination de la demande et en interrogeant le service rpcbind sur l'hôte de destination [RFC1833]. Dans tous les cas, même en ne chiffrant pas les trois numéros, RPCSEC_GSS améliore quand même l'état de la sécurité sur ce que les services RPC existants avaient de disponible précédemment. Les mises en œuvre des nouveaux services RPC qui sont soucieux de ce risque peuvent opter pour la création d'un champ "sous-procédure" qui est inclus dans les arguments d'appel spécifiques du service.

7.2 Attaques de numéro de séquence

7.2.1 Numéros de séquence au dessus de la fenêtre

Un attaquant ne peut pas amener le serveur à relever le numéro de séquence au delà de la gamme dont le client légitime est informé (et donc monter une attaque de déni de serveur) sans construire une demande RPC qui réussisse à la vérification de somme de contrôle d'en-tête. Si le coût de vérification de la somme de contrôle d'en-tête est suffisamment élevé (selon la vitesse du processeur qui fait la somme de contrôle et le coût de l'algorithme de somme de contrôle) il est possible d'envisager une attaque de déni de service (vandalisme, sous la forme d'un gâchis de ressources de traitement) par lequel l'attaquant envoie des demandes qui sont au dessus de la fenêtre. La plus simple méthode peut être pour l'attaquant de surveiller le trafic du réseau puis de choisir un numéro de séquence qui soit bien au-dessus du numéro de séquence en cours. Puis l'attaquant peut envoyer des demandes boguées en utilisant le numéro de séquence au dessus de la fenêtre.

7.2.2 Numéros de séquence dans ou au dessous de la fenêtre

Si l'attaquant envoie des demandes qui sont dans ou en dessous de la fenêtre, même si la somme de contrôle d'en-tête est alors bien vérifiée, le serveur va éliminer en silence les demandes parce qu'il supposera qu'il a déjà traité la demande. Dans

ce cas, un serveur peut optimiser en sautant la vérification de somme de contrôle d'en-tête si le numéro de séquence est en dessous de la fenêtre, ou si il est dans la fenêtre, ne pas tenter la vérification de somme de contrôle si le numéro de séquence a déjà été vu.

7.3 Attaques de vol de message

La présente proposition ne traite pas des attaques où un attaquant peut bloquer ou voler des messages sans être détecté par le serveur. Mettre en œuvre un telle protection serait équivalent à supposer un état dans le service RPC. RPCSEC_GSS n'empire pas cette situation.

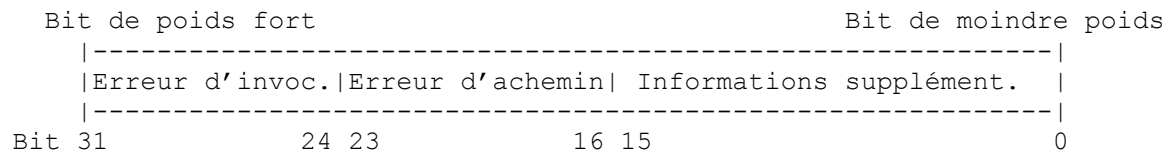
Appendice A Codes d'état majeur de GSS-API

La définition de GSS-API [RFC2078] ne comporte pas de valeurs numériques pour les divers codes d'états majeurs de GSS-API. Il est prévu que ceci soit traité dans une future RFC. Jusque là, le présent appendice définit les valeurs pour chaque code d'état majeur de GSS-API énumérés dans la définition de GSS-API. Si à l'avenir, la définition de GSS-API donne des valeurs pour les codes qui sont différents de ce qui suit, les mises en œuvre de RPCSEC_GSS seront obligées de les transposer en les valeurs définies ci-dessous. Si à l'avenir la définition de GSS-API fixe des codes d'état supplémentaires en plus de ceux définis ci-dessous, la définition de RPCSEC_GSS englobera ces valeurs supplémentaires.

Voici les définitions de chaque état majeur GSS_S_* que les mises en œuvre de RPCSEC_GSS peuvent s'attendre à trouver dans le champ gss_major de rpc_gss_init_res. Ces définitions ne sont pas dans la forme du langage de description de RPC. Les numéros sont en base 16 (hexadécimal) :

GSS_S_COMPLETE	0x00000000
GSS_S_CONTINUE_NEEDED	0x00000001
GSS_S_DUPLICATE_TOKEN	0x00000002
GSS_S_OLD_TOKEN	0x00000004
GSS_S_UNSEQ_TOKEN	0x00000008
GSS_S_GAP_TOKEN	0x00000010
GSS_S_BAD_MECH	0x00010000
GSS_S_BAD_NAME	0x00020000
GSS_S_BAD_NAMETYPE	0x00030000
GSS_S_BAD_BINDINGS	0x00040000
GSS_S_BAD_STATUS	0x00050000
GSS_S_BAD_MIC	0x00060000
GSS_S_BAD_SIG	0x00060000
GSS_S_NO_CRED	0x00070000
GSS_S_NO_CONTEXT	0x00080000
GSS_S_DEFECTIVE_TOKEN	0x00090000
GSS_S_DEFECTIVE_CREDENTIAL	0x000a0000
GSS_S_CREDENTIALS_EXPIRED	0x000b0000
GSS_S_CONTEXT_EXPIRED	0x000c0000
GSS_S_FAILURE	0x000d0000
GSS_S_BAD_QOP	0x000e0000
GSS_S_UNAUTHORIZED	0x000f0000
GSS_S_UNAVAILABLE	0x00100000
GSS_S_DUPLICATE_ELEMENT	0x00110000
GSS_S_NAME_NOT_MN	0x00120000
GSS_S_CALL_INACCESSIBLE_READ	0x01000000
GSS_S_CALL_INACCESSIBLE_WRITE	0x02000000
GSS_S_CALL_BAD_STRUCTURE	0x03000000

Noter que l'état majeur GSS-API se partage en trois champs comme suit :



Un seul état dans le champ Erreur d'invocation peut être logiquement OUixé avec un état dans le champ Erreur d'acheminement qui à son tour peut être logiquement OUixé avec zéro, un ou plusieurs états dans le champ Informations supplémentaires. Si l'état majeur résultant a une erreur d'invocation et une erreur de sous-programme différente de zéro, l'opération GSS-API applicable a alors échoué. Pour les besoins de RPCSEC_GSS, cela signifie que l'invocation GSS_Accept_sec_context() exécutée par le serveur a échoué.

Si l'état majeur est égal à GSS_S_COMPLETE, cela indique alors l'absence de toutes informations d'erreur ou supplémentaires.

La signification de la plupart des états de GSS_S_* est définie dans la définition de GSS-API, à l'exception de :

GSS_S_BAD_MIC	Ce code a la même signification que GSS_S_BAD_SIG.
GSS_S_CALL_INACCESSIBLE_READ	Un paramètre d'entrée obligé n'a pas pu être lu.
GSS_S_CALL_INACCESSIBLE_WRITE	Un paramètre d'entrée obligé n'a pas pu être écrit.
GSS_S_CALL_BAD_STRUCTURE	Un paramètre était mal formé.

Remerciements

Beaucoup du protocole se fonde sur la nuance de sécurité AUTH_GSSAPI développée par Open Vision Technologies [Jaspan]. En particulier, nous remercions Barry Jaspan, Marc Horowitz, John Linn, et Ellen McDermott. Raj Srinivasan qui ont conçu RPCSEC_GSS [Eisler] avec des apports de Mike Eisler. Raj, Roland Schemers, Lin Ling, et Alex Chiu ont contribué à la mise en œuvre par Sun Microsystems de RPCSEC_GSS. Brent Callaghan, Marc Horowitz, Barry Jaspan, John Linn, Hilarie Orman, Martin Rex, Ted Ts'o, et John Wroclawski ont analysé la spécification et donné de précieux retours. Steve Nahm et Kathy Slattery ont relu divers projets de la présente spécification. Beaucoup du contenu de l'Appendice A a été tiré du travail en cours de John Wray sur les liens C de GSS-API Version 2.

Références

[Eisler] Eisler, M., Schemers, R., and Srinivasan, R. (1996). "Security Mechanism Independence in ONC RPC," Proceedings of the Sixth Annual USENIX Security Symposium, pp. 51-65.

[Jaspan] Jaspan, B. (1995). "GSS-API Security for ONC RPC," `95 Proceedings of The Internet Society Symposium on Network and Distributed System Security, pp. 144- 151.

[RFC1831] R. Srinivasan, "RPC : Spécification de la version 2 du protocole d'appel de procédure à distance", août 1995. *(P.S.) (Obsolète, voir* RFC5531)

[RFC1832] R. Srinivasan, "XDR : norme de représentation des données externes", août 1995. *(Obsolète, voir* RFC4506) *(D.S.)*

[RFC1833] R. Srinivasan, "Protocoles de liaison pour RPC ONC version 2", août 1995. *(P.S.) (MàJ par* RFC5665*)*

[RFC2078] J. Linn, "Interface générique de programme d'application de service de sécurité, version 2", janvier 1997. *(Rendue obsolète par la* RFC2743*.)*

Adresse des auteurs

Michael Eisler
Sun Microsystems, Inc.
M/S UCOS03
2550 Garcia Avenue
Mountain View, CA 94043
téléphone : +1 (719) 599-9026
mél : mre@eng.sun.com

Alex Chiu
Sun Microsystems, Inc.
M/S UMPK17-203
2550 Garcia Avenue
Mountain View, CA 94043
téléphone :+1 (415) 786-6465
mél : hacker@eng.sun.com

Lin Ling
Sun Microsystems, Inc.
M/S UMPK17-201
2550 Garcia Avenue
Mountain View, CA 94043
téléphone :+1 (415) 786-5084
mél : lling@eng.sun.com