Groupe de travail Réseau
Request for Comments : 4462
Catégorie : Sur la voie de la normalisation

Traduction Claude Brière de L'Isle

J. Hutzelman, CMU
J. Salowey, Cisco Systems
J. Galbraith, Van Dyke Technologies, Inc.
V. Welch, U Chicago / ANL
mai 2006

Authentification et échange de clés d'interface de programme d'application de service de sécurité générique (GSS-API) pour le protocole Secure Shell (SSH)

Résumé
Le protocole Secure Shell (SSH) est un protocole de connexion sûre à distance et autres services réseau sûrs sur un réseau non sûr. L'interface de programme d'application de service générique de sécurité (GSS-API, *Generic Security Service Application Program Interface*) fournit des services de sécurité aux appelants d'une façon indépendante du mécanisme.

Le présent mémoire décrit les méthodes d'utilisation de GSS-API pour l'authentification et l'échange de clés dans SSH. Il définit une méthode d'authentification d'utilisateur SSH qui utilise un mécanisme de GSS-API spécifié pour authentifier un utilisateur, et une famille de méthodes d'échange de clés SSH qui utilise GSS-API pour authentifier un échange de clés Diffie-Hellman.

Le présent mémoire définit aussi un nouvel algorithme de clé publique d'hôte qui peut être utilisé quand aucune opération n'est nécessaire en utilisant la clé publique d'un hôte, et une nouvelle méthode d'authentification d'utilisateur qui permet qu'un nom d'autorisation soit employé en conjonction avec toute authentification qui s'est déjà produite comme effet collatéral de l'échange de clé fondé sur GSS-API.

Table des matières

1. Introduction

Le présent document décrit les méthodes utilisées pour effectuer l'échange de clés et l'authentification d'utilisateur dans le protocole Secure Shell en utilisant la GSS-API. Pour ce faire, il définit une famille de méthodes d'échange de clés, deux méthodes d'authentification d'utilisateur et un nouvel algorithme de clé d'hôte. Ces définitions permettent à tout mécanisme de GSS-API d'être utilisé avec le protocole Secure Shell.

Le présent document ne devrait être lu qu'après les documents qui décrivent l'architecture du protocole SSH [RFC4251], le protocole de couche transport [RFC4253], et le protocole d'authentification d'utilisateur [RFC4252]. Ce document utilise librement la terminologie et notation du document d'architecture sans autre référence ou explication.

1.1 Terminologie de SSH

Les types de données utilisés dans les paquets sont définis dans le document d'architecture SSH [RFC4251]. Il est particulièrement important de noter la définition des chaînes qui permettent un contenu binaire.

Le paquet SSH_MSG_USERAUTH_REQUEST se réfère à un service ; ce nom de service est un nom de service SSH et n'a pas de relation avec les noms de service GSS-API. Actuellement, le seul nom de service défini est "ssh-connection", qui se réfère au protocole de connexion SSH [RFC4254].

1.2 Mots clés

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" en majuscules dans ce document sont à interpréter comme décrit dans le BCP 14, [RFC2119].

2. Échange de clé Diffie-Hellman authentifié par GSS-API

Cette Section définit une classe de méthodes d'échange de clés qui combine l'échange de clés Diffie-Hellman de la Section 8 de la [RFC4253] avec l'authentification mutuelle qui utilise GSS-API.

Comme les méthodes d'échange de clés GSS-API décrites dans cette section n'exigent pas l'utilisation de signature de clé publique ou d'algorithmes de chiffrement, elles PEUVENT être utilisées avec tout algorithme de clé d'hôte, incluant l'algorithme "nul" décrit à la Section 5.

2.1 Échange de clé générique GSS-API

Les symboles suivants sont utilisés dans cette description :
C : le client, et S est le serveur.
p : un grand nombre premier sûr, g est un générateur pour un sous groupe de GF(p), et q est l'ordre du sous groupe.
V_S : la chaîne de version de S, et V_C est la chaîne de version de C.
I_C : le message KEXINIT de C, et I_S est le message KEXINIT de S.

1. C génère un nombre aléatoire x (1 < x < q) et calcule e = g^x mod p.

2. C invoque GSS_Init_sec_context(), en utilisant le plus récent jeton de réponse reçu de S durant cet échange, si il en est. Pour cet appel, le client DOIT régler mutual_req_flag à "vrai" pour demander que l'authentification mutuelle soit effectuée. Il DOIT aussi régler integ_req_flag à "vrai" pour demander que la protection de l'intégrité par message soit prise en charge pour ce contexte. De plus, deleg_req_flag PEUT être réglé à "vrai" pour demander une délégation d'accès, si elle est demandée par l'utilisateur. Comme le processus d'échange de clés n'authentifie que l'hôte, le réglage de anon_req_flag est immatériel pour ce processus. Si le client ne prend pas en charge la méthode d'authentification d'utilisateur "gssapi-keyex" décrite à la Section 4, ou n'a pas l'intention d'utiliser cette méthode en conjonction avec le contexte GSS-API établi durant l'échange de clés, alors anon_req_flag DEVRAIT être réglé à "vrai". Autrement, ce fanion PEUT être réglé à vrai si le client souhaite cacher son identité. Comme le processus d'échange de clés va impliquer l'échange d'un seul jeton une fois que le contexte a été établi, il n'est pas nécessaire que le contexte GSS-API prenne en charge la détection des jetons répétés ou hors séquence. Donc, replay_det_req_flag et sequence_req_flag n'ont pas besoin d'être établis pour ce processus. Ces fanions DEVRAIT être réglés à "faux".
   * Si le code résultant major_status est GSS_S_COMPLETE et si le fanion mutual_state n'est pas vrai, l'authentification mutuelle n'a alors pas été établie, et l'échange de clés DOIT échouer.
   * Si le code résultant major_status est GSS_S_COMPLETE et si le fanion integ_avail n'est pas vrai, la protection d'intégrité par message n'est alors pas disponible, et l'échange de clés DOIT échouer.
   * Si le code résultant major_status est GSS_S_COMPLETE et si les deux fanions mutual_state et integ_avail sont vrais, le jeton de résultat est envoyé à S.
   * Si le code résultant major_status est GSS_S_CONTINUE_NEEDED, le jeton de sortie output_token est envoyé à S, qui va répondre avec un nouveau jeton à fournir à GSS_Init_sec_context().
   * Le client DOIT aussi inclure "e" avec le premier message qu'il envoie au serveur durant ce processus ; si le serveur reçoit plus d'un "e" ou aucun, l'échange de clés échoue.
   * C'est une erreur si l'appel ne produit pas un jeton de longueur non zéro à envoyer au serveur. Dans ce cas, l'échange de clés DOIT échouer.

3. S invoque GSS_Accept_sec_context(), en utilisant le jeton reçu de C.
   * Si le code résultant major_status est GSS_S_COMPLETE et si le fanion mutual_state n'est pas vrai, l'authentification mutuelle n'a pas été établie, et l'échange de clés DOIT échouer.
   * Si le code résultant major_status est GSS_S_COMPLETE et si le fanion integ_avail n'est pas vrai, la protection d'intégrité par message n'est alors pas disponible, et l'échange de clés DOIT échouer.
   * Si le code résultant major_status est GSS_S_COMPLETE et si les deux fanions mutual_state et integ_avail sont vrais, le contexte de sécurité a alors été établi, et le traitement continue à l'étape 4.
   * Si le code résultant major_status est GSS_S_CONTINUE_NEEDED, le jeton de sortie est alors envoyé à C, et le traitement continue à l'étape 2.
   * Si le code résultant major_status est GSS_S_COMPLETE, mais si un jeton de réponse de longueur non zéro est retourné, ce jeton est alors envoyé au client.

4. S génère un nombre aléatoire y (0 < y < q) et calcule f = g^y mod p. Il calcule K = e ^ y mod p, et H = hash(V_C || V_S || I_C || I_S || K_S || e || f || K). Il invoque alors GSS_GetMIC() pour obtenir un code d'intégrité de message GSS-API pour H. S envoie alors f et le code d'intégrité de message (MIC, *message integrity code*) à C.

5. Cette étape n'est effectuée que (1) si l'appel final du serveur à GSS_Accept_sec_context() a produit un jeton de réponse final de longueur non zéro à envoyer au client et (2) si aucun précédent appel du client à GSS_Init_sec_context() n'a résulté en un major_status de GSS_S_COMPLETE. Dans ces conditions, le client ne fait pas d'autre appel à GSS_Init_sec_context() pour traiter le jeton de réponse final. Cet appel est fait exactement comme décrit ci-dessus. Cependant, si le major_status résultant est autre chose que GSS_S_COMPLETE, ou si un jeton de longueur non zéro est retourné, c'est une erreur et l'échange de clés DOIT échouer.

6. C calcule K = f^x mod p, et H = hash(V_C || V_S || I_C || I_S || K_S || e || f || K). Il invoque alors GSS_VerifyMIC() pour vérifier que le MIC envoyé par S correspond à H. Si la vérification du MIC échoue, l'échange de clés DOIT échouer.

L'un et l'autre côté NE DOIVENT PAS envoyer ou accepter de valeur de e ou f qui ne soient pas dans la gamme [1, p-1]. Si cette condition est violée, l'échange de clés échoue.

Si une invocation de GSS_Init_sec_context() ou GSS_Accept_sec_context() retourne un major_status autre que GSS_S_COMPLETE ou GSS_S_CONTINUE_NEEDED, ou si tout autre invocation de GSS-API retourne un major_status autre que GSS_S_COMPLETE, l'échange de clés échoue. Dans ce cas, plusieurs mécanismes sont disponibles pour communiquer les informations d'erreur à l'homologue avant de terminer la connexion comme exigé par la [RFC4253] :

- o Si l'échange de clés échoue à cause d'une erreur de GSS-API sur le serveur (incluant des erreurs retournées par GSS_Accept_sec_context()) le serveur PEUT envoyer un message informant le client des détails de l'erreur. Dans ce cas, si un jeton d'erreur est aussi envoyé (voir ci-dessous) ce message DOIT alors être envoyé avant le jeton d'erreur.
- o Si l'échange de clés échoue à cause d'une erreur GSS-API retournée de l'appel du serveur à GSS_Accept_sec_context() et si un "jeton d'erreur" est aussi retourné, le serveur DEVRAIT alors envoyer le jeton d'erreur au client pour permettre l'achèvement de l'échange de sécurité GSS.
- o Si l'échange de clés échoue à cause d'une erreur de GSS-API retournée de l'appel du client à GSS_Init_sec_context(), et si un "jeton d'erreur" est aussi retourné, le client DEVRAIT alors envoyer le jeton d'erreur au serveur pour permettre l'achèvement de l'échange de sécurité GSS.

Comme noté à la Section 9, il peut être souhaitable selon la politique de sécurité du site d'obscurcir les informations sur la nature précise de l'erreur ; donc, il est RECOMMANDÉ que les mises en œuvre fournissent une méthode pour supprimer ces messages au titre de la politique.

Ceci est mis en œuvre avec les messages suivants. L'algorithme de hachage pour calculer le hachage de l'échange est défini par le nom de la méthode, et est appelé HASH. Le groupe utilisé pour l'échange de clés Diffie-Hellman et le mécanisme GSS-API sous-jacent sont aussi définis par le nom de la méthode.

Après le premier appel du client à GSS_Init_sec_context(), il envoie ce qui suit :

byte SSH_MSG_KEXGSS_INIT
string output_token (de GSS_Init_sec_context())
mpint e

À réception du message SSH_MSG_KEXGSS_INIT, le serveur PEUT envoyer le message suivant, avant tout autre message, pour informer le client de sa clé d'hôte :

byte SSH_MSG_KEXGSS_HOSTKEY
string clé publique d'hôte du serveur et certificats (K_S)

Comme cette méthode d'échange de clés n'exige pas que la clé d'hôte soit utilisée pour une opération de chiffrements, ce message est FACULTATIF. Si l'algorithme de clé d'hôte "nul" décrit à la Section 5 est utilisé, ce message NE DOIT PAS être envoyé. Si ce message est envoyé, la ou les clés publiques d'hôte du serveur et/ou le ou les certificats dans ce message sont codés comme une seule chaîne, dans le format spécifié par le type de clé publique utilisée (voir la [RFC4253], paragraphe 6.6).

Dans les déploiements traditionnels de SSH, les clés d'hôte sont normalement supposées ne changer que rarement, et il n'y a souvent aucun mécanisme pour valider les clés d'hôte qui ne sont pas déjà connues du client. Par suite, l'utilisation d'une nouvelle clé d'hôte par un hôte déjà connu est généralement considérée comme l'indication d'une possible attaque par interposition, et les clients affichent souvent de forts avertissement et/ou interrompent la connexion dans de tels cas.

À l'opposé, quand un échange de clés fondé sur GSS-API est utilisé, les clés d'hôte envoyées via le message SSH_MSG_KEXGSS_HOSTKEY sont authentifiés au titre de l'échange de clés GSS-API, même quand elles ne sont pas connues à l'avance par le client. De plus, dans les environnements dans lesquels l'échange de clés fondé sur GSS-API est utilisé lourdement, il est possible et même probable que les clés d'hôte vont changer beaucoup plus fréquemment et/ou sans avertissement préalable.

Donc, quand une nouvelle clé pour un hôte déjà connu est reçue via le message SSH_MSG_KEXGSS_HOSTKEY, les clients NE DEVRAIENT PAS produire de forts avertissements ou interrompre la connexion, pourvu que l'échange de clés fondé sur GSS-API réussisse.

Afin de faciliter le re-échange de clés après l'arrivée à expiration des accréditifs GSS-API de l'utilisateur, les mises en œuvre de client DEVRAIENT mémoriser les clés d'hôte reçues via SSH_MSG_KEXGSS_HOSTKEY pour la durée de la session, même quand de telles clés ne sont pas mémorisées pour une utilisation à long terme.

Chaque fois que l'invocation du serveur à GSS_Accept_sec_context() retourne un code major_status de GSS_S_CONTINUE_NEEDED, il envoie la réponse suivante au client :

byte SSH_MSG_KEXGSS_CONTINUE
string output_token (de GSS_Accept_sec_context())

Si le client reçoit ce message après qu'une invocation à GSS_Init_sec_context() a retourné un code major_status de

GSS_S_COMPLETE, une erreur de protocole s'est produite et l'échange de clés DOIT échouer.

Chaque fois que le client reçoit le message décrit ci-dessus, il fait une autre invocation de GSS_Init_sec_context(). Il envoie alors ce qui suit :

byte SSH_MSG_KEXGSS_CONTINUE
string output_token (de GSS_Init_sec_context())

Le serveur et le client continuent d'échanger ces deux messages tant que les invocations du serveur à GSS_Accept_sec_context() résultent en des codes major_status de GSS_S_CONTINUE_NEEDED. Quand un appel résulte en un code major_status de GSS_S_COMPLETE, il envoie un des deux messages finaux.

Si l'appel final du serveur à GSS_Accept_sec_context() (résultant en un code major_status de GSS_S_COMPLETE) retourne un jeton de longueur non zéro à envoyer au client, il envoie ce qui suit :

byte SSH_MSG_KEXGSS_COMPLETE
mpint f
string per_msg_token (MIC de H)
boolean VRAI
string output_token (de GSS_Accept_sec_context())

Si le client reçoit ce message après qu'une invocation à GSS_Init_sec_context() a retourné un code major_status de GSS_S_COMPLETE, une erreur de protocole s'est produite et l'échange de clés DOIT échouer.

Si l'appel final du serveur à GSS_Accept_sec_context() (résultant en un code major_status de GSS_S_COMPLETE) retourne un jeton de longueur zéro ou pas de jeton du tout, il envoie ce qui suit :

byte SSH_MSG_KEXGSS_COMPLETE
mpint f
string per_msg_token (MIC de H)
boolean FAUX

Si le client reçoit ce message alors qu'aucun appel à GSS_Init_sec_context() n'a encore résulté en un code major_status de GSS_S_COMPLETE, une erreur de protocole s'est produite et l'échange de clés DOIT échouer.

Si l'invocation du client à GSS_Init_sec_context() ou l'appel du serveur à GSS_Accept_sec_context() retourne un état d'erreur et produit un jeton de résultat (appelé un "jeton d'erreur") alors ce qui suit DEVRAIT être envoyé pour porter les informations d'erreur à l'homologue :

byte SSH_MSG_KEXGSS_CONTINUE
string error_token

Si un serveur envoie à la fois ce message et un message SSH_MSG_KEXGSS_ERROR, le message SSH_MSG_KEXGSS_ERROR DOIT être envoyé d'abord, pour permettre aux clients d'enregistrer et/ou afficher les informations d'erreur avant de traiter le jeton d'erreur. Ceci est important parce qu'un client qui traite un jeton d'erreur va probablement se déconnecter sans lire aucun autre message.

Dans le cas d'une erreur GSS-API sur le serveur, le serveur PEUT envoyer le message suivant avant de terminer la connexion :

byte SSH_MSG_KEXGSS_ERROR
uint32 major_status
uint32 minor_status
string message
string étiquette de langue

Le texte du message DOIT être codé en UTF-8, décrit dans la [RFC3629]. Les étiquettes de langue sont celles décrites dans la [RFC3066]. Noter que le texte du message peut contenir plusieurs lignes séparées par des séquence de saut à la ligne retour chariot (CRLF). Les développeurs d'applications devraient prendre cela en compte pour l'affichage de ces messages.

Le hachage H est calculé comme le hachage HASH de l'enchaînement de ce qui suit :
string V_C, chaîne de la version du client (CR, NL exclus)

string V_S, chaîne de la version du serveur (CR, NL exclus)
string I_C, charge utile du SSH_MSG_KEXINIT du client
string I_S, charge utile du SSH_MSG_KEXINIT du serveur
string K_S, clé de l'hôte
mpint e, valeur d'échange envoyée par le client
mpint f, valeur d'échange envoyée par le serveur
mpint K, secret partagé

Cette valeur est appelée le hachage d'échange, et elle est utilisée pour authentifier l'échange de clés. Le hachage d'échange DEVRAIT être gardé secret. Si aucun message SSH_MSG_KEXGSS_HOSTKEY n'a été envoyé par le serveur ou reçu par le client, la chaîne vide est alors utilisée à la place de K_S pour calculer le hachage d'échange.

L'appel GSS_GetMIC DOIT être appliqué sur H, et non sur les données d'origine.

2.2 Échange de groupe

Ce paragraphe décrit une modification de l'échange de clés Diffie-Hellman générique authentifié par GSS-API pour permettre la négociation du groupe à utiliser, avec une méthode fondée sur celle décrite dans la [RFC4419].

Le serveur tient une liste de nombres premiers sûrs et des paramètres correspondants dans laquelle il peut faire son choix. Ils sont choisis comme décrit dans la Section 3 de la [RFC4419]. Le client demande un module au serveur, indiquant les tailles minimum, maximum, et préférées ; le serveur répond avec un module et un générateur convenables. L'échange se poursuit alors comme décrit au paragraphe 2.1 ci-dessus.

Cette description utilise les symboles suivants, en plus de ceux définis précédemment :
n : est la taille du module p en bits que le client aimerait recevoir du serveur
min et max : sont les tailles minimales et maximales de p en bits qui sont acceptables au client

1. C envoie "min || n || max" à S, pour indiquer la taille de groupe minimale acceptable, la taille préférée du groupe, et la taille maximale du groupe en bits que le client va accepter.

2. S trouve un groupe qui correspond bien à la demande du client, et envoie "p || g" à C.

3. L'échange s'effectue comme décrit au paragraphe 2.1, en commençant par l'étape 1, sauf que le hachage d'échange est calculé comme décrit ci-dessous.

Les serveurs et les clients DEVRAIENT accepter les groupes avec une longueur de module de k bits, où $1024 \leq k \leq 8192$. Les valeurs recommandées pour min et max sont respectivement 1024 et 8192.

Ceci est mis en œuvre en utilisant les messages suivants, en plus de ceux décrits précédemment :

D'abord, le client envoie :

byte SSH_MSG_KEXGSS_GROUPREQ
uint32 min, taille minimale en bits d'un groupe acceptable
uint32 n, taille préférée en bits du groupe que le serveur devrait envoyer
uint32 max, taille maximale en bits d'un groupe acceptable

Le serveur répond avec :

byte SSH_MSG_KEXGSS_GROUP
mpint p, nombre premier sûr
mpint g, générateur de sous groupe dans GF(p)

Ceci est suivi par l'échange de messages décrit au paragraphe 2.1, sauf que le hachage d'échange H est calculé comme le hachage HASH de l’enchaînement de ce qui suit :

string V_C, chaîne de version du client (CR, NL exclus)
string V_S, chaîne de version du serveur (CR, NL exclus)
string I_C, charge utile du SSH_MSG_KEXINIT du client
string I_S, charge utile du SSH_MSG_KEXINIT du serveur

string K_S, clé d'hôte
uint32 min, taille minimale en bits d'un groupe acceptable
uint32 n, taille préférée en bits du groupe que le serveur devrait envoyer
uint32 max, taille maximale en bits d'un groupe acceptable
mpint p, nombre premier sûr
mpint g, générateur pour le sous groupe dans GF(p)
mpint e, valeur d'échange envoyée par le client
mpint f, valeur d'échange envoyée par le serveur
mpint K, secret partagé

2.3 gss-group1-sha1-*

Chacune de ces méthodes spécifie un échange de clés Diffie-Hellman authentifié par GSS-API comme décrit au paragraphe 2.1 avec SHA-1 comme HASH, et le groupe défini au paragraphe 8.1 de la [RFC4253]. Le nom de méthode pour chaque méthode est l'enchaînement de la chaîne "gss-group1-sha1-" avec le codage en Base64 du hachage MD5 [RFC1321] du codage en DER ASN.1 (DER, *Distinguished Encoding Rules*) [ASN1] de l'identifiant d'objet (OID, *Object Identifier*) du mécanisme GSS-API sous-jacent. Le codage en Base64 est décrit au paragraphe 6.8 de la [RFC2045].

Chacune de ces méthodes d'échange de clés est implicitement enregistrée par cette spécification. L'IESG est considéré comme étant propriétaire de toutes ces méthodes d'échange de clés ; cela N'IMPLIQUE PAS que l'IESG est considéré comme étant propriétaire des mécanismes GSS-API sous-jacents.

2.4 gss-group14-sha1-*

Chacune de ces méthodes spécifie un échange de clés Diffie-Hellman authentifié par GSS-API comme décrit au paragraphe 2.1 avec SHA-1 comme HASH, et le groupe défini au paragraphe 8.2 de la [RFC4253]. Le nom de méthode pour chaque méthode est l'enchaînement de la chaîne "gss-group14-sha1-" avec le codage en Base64 du hachage MD5 [RFC1321] du codage en DER ASN.1 [ASN1] de l'OID du mécanisme GSS-API sous-jacent. Le codage en Base64 est décrit au paragraphe 6.8 de la [RFC2045].

Chacune de ces méthodes d'échange de clés est implicitement enregistrée par cette spécification. L'IESG est considéré comme étant propriétaire de toutes ces méthodes d'échange de clés ; cela N'IMPLIQUE PAS que l'IESG est considéré comme étant propriétaire des mécanismes GSS-API sous-jacents.

2.5 gss-gex-sha1-*

Chacune de ces méthodes spécifie un échange de clés Diffie-Hellman authentifié par GSS-API comme décrit au paragraphe 2.2 avec SHA-1 comme HASH. Le nom de méthode pour chaque méthode est l'enchaînement de la chaîne "gss-gex-sha1-" avec le codage en Base64 du hachage MD5 [RFC1321] du codage en DER ASN.1 [ASN1] de l'OID du mécanisme GSS-API sous-jacent. Le codage en Base64 est décrit au paragraphe 6.8 de la [RFC2045].

Chacune de ces méthodes d'échange de clés est implicitement enregistrée par cette spécification. L'IESG est considéré comme étant propriétaire de toutes ces méthodes d'échange de clés ; cela N'IMPLIQUE PAS que l'IESG est considéré comme étant propriétaire des mécanismes GSS-API sous-jacents.

2.6 Autres méthodes d'échange de clé GSS-API

Les noms de méthode d'échange de clés commençant pas "gss-" sont réservés pour les méthodes d'échange de clés conformes au présent document ; en particulier, pour les méthodes qui utilisent l'algorithme d'échange de clés Diffie-Hellman authentifié par GSS-API décrit au paragraphe 2.1, incluant toutes les futures méthodes qui utilisent des groupes et/ou fonctions de hachage différents. L'intention est que les noms de chacune de ces futures méthodes soient définis d'une manière similaire à celle utilisée au paragraphe 2.3.

3. Authentification d'utilisateur GSS-API

Cette section décrit une méthode d'authentification d'utilisateur d'utilité générale fondée sur la [RFC2743]. Elle est destinée à fonctionner sur le protocole SSH d'authentification d'utilisateur [RFC4252].

Le nom de la méthode d'authentification pour ce protocole est "gssapi-with-mic".

3.1 Vue d'ensemble de l'authentification GSS-API

L'authentification GSS-API doit maintenir un contexte. L'authentification commence quand le client envoie une SSH_MSG_USERAUTH_REQUEST, qui spécifie les OID de mécanisme que le client prend en charge.

Si le serveur prend en charge un des OID de mécanisme demandés, le serveur envoie un message SSH_MSG_USERAUTH_GSSAPI_RESPONSE contenant l'OID de mécanisme.

Après que le client a reçu SSH_MSG_USERAUTH_GSSAPI_RESPONSE, le client et le serveur échangent des paquets SSH_MSG_USERAUTH_GSSAPI_TOKEN jusqu'à ce que le mécanisme d'authentification réussisse ou échoue.

Si à tout moment durant l'échange, le client envoie un nouveau paquet SSH_MSG_USERAUTH_REQUEST, le contexte GSS-API est complètement éliminé et détruit, et toute autre authentification GSS-API DOIT recommencer depuis le début.

Si l'authentification réussit et si un nom d'utilisateur non vide est présenté par le client, la mise en œuvre de serveur SSH vérifie que le nom d'utilisateur est autorisé sur la base des accréditifs échangés dans l'échange GSS-API. Si le nom d'utilisateur n'est pas autorisé, l'authentification DOIT alors échouer.

3.2 Initiation de l'authentification GSS-API

La méthode d'authentification GSS-API est initiée quand le client envoie une SSH_MSG_USERAUTH_REQUEST :

byte SSH_MSG_USERAUTH_REQUEST
string nom d'utilisateur (codé en ISO-10646 UTF-8)
string nom de service (en US-ASCII)
string "gssapi-with-mic" (nom de méthode en US-ASCII)
uint32 n, nombre d'OID de mécanisme que le client supporte
string[n] OID de mécanisme

Les OID de mécanisme sont codés conformément aux règles de codage distinctives ASN.1 (DER), comme décrit dans [ASN1] et au paragraphe 3.1 de la [RFC2743]. Les OID de mécanisme DOIVENT être énumérés dans l'ordre de préférence, et le serveur doit choisir le premier OID de mécanisme qu'il prend en charge sur la liste.

Le client DEVRAIT envoyer seulement les OID de mécanisme GSS-API qui sont de même priorité, par rapport aux méthodes d'authentification non GSS-API. Autrement, les méthodes d'authentification peuvent être exécutées sans ordre. Donc, le client peut envoyer d'abord une SSH_MSG_USERAUTH_REQUEST pour un mécanisme GSS-API, puis essayer l'authentification à clé publique, et ensuite essayer un autre mécanisme GSS-API.

Si le serveur ne prend en charge aucun des OID spécifiés, le serveur DOIT faire échouer la demande en envoyant un paquet SSH_MSG_USERAUTH_FAILURE.

Le nom d'utilisateur peut être une chaîne vide si il peut être déduit du résultat de l'authentification GSS-API. Si le nom d'utilisateur n'est pas vide, et si l'utilisateur demandé n'existe pas, le serveur PEUT déconnecter ou PEUT envoyer une fausse liste d'authentifications acceptables mais n'en accepter aucune. Cela permet au serveur d'éviter de divulguer des informations sur les comptes qui existent. Dans tous les cas, si l'utilisateur n'existe pas, la demande d'authentification NE DOIT PAS être acceptée.

Noter que la valeur de "nom d'utilisateur" est codée en ISO-10646 UTF-8. Il appartient au serveur de définir comment il interprète le nom d'utilisateur et détermine si le client est autorisé sur la base de ses accréditifs GSS-API. En particulier, le codage utilisé par le système pour les noms d'utilisateur est l'affaire de la mise en œuvre de serveur SSH. Cependant, si le client lit le nom d'utilisateur dans un autre codage (par exemple, ISO 8859-1 - ISO Latin1) il DOIT convertir le nom d'utilisateur en ISO-10646 UTF-8 avant de la transmettre, et le serveur DOIT convertir le nom d'utilisateur en le codage utilisé sur ce système pour les noms d'utilisateur.

Toute normalisation ou autre préparation des noms est faite par le serveur SSH sur la base des exigences du système, et sort du domaine d'application de SSH. Les mises en œuvre de SSH qui tiennent des bases de données privées d'utilisateurs DEVRAIENT préparer les noms d'utilisateur comme décrit dans la [RFC4013].

Le client PEUT à tout moment continuer avec un nouveau message SSH_MSG_USERAUTH_REQUEST, et dans ce cas le serveur DOIT abandonner la tentative d'authentification précédente et continuer avec la nouvelle.

3.3 Réponse initiale du serveur

Le serveur répond à la SSH_MSG_USERAUTH_REQUEST par SSH_MSG_USERAUTH_FAILURE si aucun des mécanismes n'est pris en charge ou avec une SSH_MSG_USERAUTH_GSSAPI_RESPONSE comme suit :

byte SSH_MSG_USERAUTH_GSSAPI_RESPONSE
string OID du mécanisme choisi

L'OID de mécanisme doit être un des OID envoyés par le client dans le paquet SSH_MSG_USERAUTH_REQUEST.

3.4 Session GSS-API

Une fois que l'OID de mécanisme a été choisi, le client va alors initier un échange d'une ou plusieurs paires de paquets SSH_MSG_USERAUTH_GSSAPI_TOKEN. Ces paquets contiennent les jetons produits à partir des invocations de 'GSS_Init_sec_context()' et 'GSS_Accept_sec_context()'. Le nombre réel de paquets échangés est déterminé par le mécanisme GSS-API sous-jacent.

byte SSH_MSG_USERAUTH_GSSAPI_TOKEN
string données retournées de GSS_Init_sec_context() ou GSS_Accept_sec_context()

Si une erreur se produit durant cet échange sur le côté serveur, le serveur peut terminer la méthode en envoyant un paquet SSH_MSG_USERAUTH_FAILURE. Si une erreur se produit du côté client, le client peut terminer la méthode en envoyant un nouveau paquet SSH_MSG_USERAUTH_REQUEST.

Lorsque il invoque GSS_Init_sec_context(), le client DOIT régler integ_req_flag à "vrai" pour demander que la protection d'intégrité par message soit prise en charge pour ce contexte. De plus, le fanion deleg_req_flag PEUT être réglé à "vrai" pour demander une délégation d'accès, si c'est demandé par l'utilisateur.

Comme par nature le processus d'authentification d'utilisateur n'authentifie que le client, l'établissement du fanion mutual_req_flag n'est pas nécessaire pour ce processus. Ce fanion DEVRAIT être réglé à "faux".

Comme le processus d'authentification d'utilisateur va impliquer l'échange d'un seul jeton une fois que le contexte a été établi, il n'est pas nécessaire que le contexte prenne en charge la détection des jetons répétés ou hors séquence. Donc, l'établissement des fanions replay_det_req_flag et sequence_req_flag n'est pas nécessaire dans ce processus. Ces fanions DEVRAIENT être réglés à "faux".

Des messages SSH_MSG_USERAUTH_GSSAPI_TOKEN supplémentaires ne sont envoyés que si et seulement si l'invocation des sous programmes GSS-API produit des jetons d'envoi de longueur non zéro.

Tout code major status autre que GSS_S_COMPLETE ou GSS_S_CONTINUE_NEEDED DEVRAIT être un échec.

3.5 Clés de chiffrement de lien

Dans certains cas, il est possible d'obtenir une amélioration de la sécurité en ne permettant l'accès que si le client envoie un code d'intégrité de message (MIC) valide liant le contexte GSS-API aux clés utilisées pour le chiffrement et la protection de l'intégrité de la session SSH. Avec ce niveau supplémentaire de protection, un attaquant "interposé" qui a convaincu un client de son authenticité ne peut alors pas relayer les messages d'authentification d'utilisateur entre le client réel et le serveur, gagnant ainsi l'accès au serveur réel. Cette protection supplémentaire est disponible quand le contexte GSS-API négocié prend en charge la protection d'intégrité par message, comme indiqué par l'établissement du fanion integ_avail sur un retour réussi de GSS_Init_sec_context() ou GSS_Accept_sec_context().

Quand l'invocation du client à GSS_Init_sec_context() retourne GSS_S_COMPLETE avec le fanion integ_avail établi, le client DOIT conclure l'échange d'authentification d'utilisateur par l'envoi du message suivant :

byte SSH_MSG_USERAUTH_GSSAPI_MIC
string MIC

Ce message DOIT n'être envoyé que si GSS_Init_sec_context() a retourné GSS_S_COMPLETE. Si un jeton est aussi retourné, le message SSH_MSG_USERAUTH_GSSAPI_TOKEN DOIT alors être envoyé avant celui-là.

Le contenu du champ MIC est obtenu par l'invocation de GSS_GetMIC() sur ce qui suit, en utilisant le contexte GSS-API qui vient d'être établi :

string identifiant de session
byte SSH_MSG_USERAUTH_REQUEST
string nom d'utilisateur
string service
string "gssapi-with-mic"

Si ce message est reçu par le serveur avant que le contexte GSS-API soit pleinement établi, le serveur DOIT faire échouer l'authentification.

Si ce message est reçu par le serveur quand le contexte GSS-API négocié ne prend pas en charge la protection d'intégrité par message, le serveur DOIT faire échouer l'authentification.

3.6 Accusé de réception du client

Certains serveurs peuvent souhaiter permettre que l'authentification d'utilisateur s'effectue même quand le contexte GSS-API négocié ne prend pas en charge la protection d'intégrité par message. Dans ce cas, il est possible au serveur de terminer avec succès la méthode complète GSS-API, alors que la dernière invocation du client à GSS_Init_sec_context() échoue. Si le serveur a simplement supposé le succès de la part du client et achevé le service d'authentification, il est possible que le client échoue à terminer la méthode d'authentification, mais ne soit pas capable de réessayer avec d'autres méthodes parce que le serveur est déjà parti. Pour se protéger contre cela, un message final est envoyé par le client pour indiquer qu'il a achevé l'authentification.

Quand l'invocation du client à GSS_Init_sec_context() retourne GSS_S_COMPLETE avec le fanion integ_avail non établi, le client DOIT conclure l'échange d'authentification d'utilisateur en envoyant le message suivant :

byte SSH_MSG_USERAUTH_GSSAPI_EXCHANGE_COMPLETE

Ce message ne DOIT être envoyé que si GSS_Init_sec_context() a retourné GSS_S_COMPLETE. Si un jeton est aussi retourné, le message SSH_MSG_USERAUTH_GSSAPI_TOKEN DOIT alors être envoyé avant celui-ci.

Si ce message est reçu par le serveur avant que le contexte GSS-API soit pleinement établi, le serveur DOIT faire échouer l'authentification.

Si ce message est reçu par le serveur quand le contexte GSS-API négocié prend en charge la protection d'intégrité par message, le serveur DOIT faire échouer l'authentification.

C'est une décision de politique du site du serveur de permettre ou non l'authentification en utilisant les mécanismes GSS-API et/ou contextes qui ne prennent pas en charge la protection d'intégrité par message. Le serveur PEUT faire échouer une authentification GSS-API avec MIC par ailleurs valide si la protection d'intégrité par message n'est pas prise en charge.

3.7 Achèvement

Comme avec toutes les méthodes d'authentification SSH, la réussite de l'achèvement est indiquée par un message SSH_MSG_USERAUTH_SUCCESS si aucune autre authentification n'est requise, ou un message SSH_MSG_USERAUTH_FAILURE avec le fanion succès partiel établi si le serveur exige plus d'authentification. Ce paquet DEVRAIT être envoyé immédiatement à la suite de la réception du paquet SSH_MSG_USERAUTH_GSSAPI_EXCHANGE_COMPLETE.

3.8 État d'erreur

Dans l'éventualité d'une erreur GSS-API sur le serveur durant l'établissement du contexte, le serveur PEUT envoyer le message suivant pour informer le client des détails de l'erreur avant d'envoyer un message SSH_MSG_USERAUTH_FAILURE :

byte SSH_MSG_USERAUTH_GSSAPI_ERROR
uint32 major_status
uint32 minor_status
string message
string étiquette de langue

Le texte du message DOIT être codé en UTF-8 comme décrit dans la [RFC3629]. Les étiquettes de langue sont celles décrites dans la [RFC3066]. Noter que le texte du message peut contenir plusieurs lignes séparées par des séquences de retour chariot-saut à la ligne (CRLF). Les développeurs d'applications devraient en tenir compte pour l'affichage de ces messages.

Les clients qui reçoivent ce message PEUVENT enregistrer les détails de l'erreur et/ou les rapporter à l'utilisateur. Tout serveur qui envoie ce message DOIT ignorer tout SSH_MSG_UNIMPLEMENTED envoyé en réponse par le client.

3.9 Jeton d'erreur

Dans l'éventualité où, durant l'établissement du contexte, l'invocation d'un client à GSS_Init_sec_context() ou d'un serveur à GSS_Accept_sec_context() retourne un jeton avec un état d'erreur, le "jeton d'erreur" résultant DEVRAIT être envoyé à l'homologue en utilisant le message suivant :

byte SSH_MSG_USERAUTH_GSSAPI_ERRTOK
string jeton d' erreur

Ce message implique que l'authentification est sur le point d'échouer, et est défini pour permettre que le jeton d'erreur soit communiqué sans perdre la synchronisation.

Quand un serveur envoie ce message, il DOIT être suivi d'un message SSH_MSG_USERAUTH_FAILURE, qui est à interpréter comme s'appliquant à la même demande d'authentification. Un client qui reçoit ce message DEVRAIT attendre le message SSH_MSG_USERAUTH_FAILURE suivant avant de commencer une autre tentative d'authentification.

Quand un client envoie ce message, il DOIT être suivi par une nouvelle demande d'authentification ou par la fermeture de la connexion. Un serveur qui reçoit ce message NE DOIT PAS envoyer un SSH_MSG_USERAUTH_FAILURE en réponse, car un tel message serait interprété par un client comme une réponse à la séquence d'authentification suivante.

Tout serveur qui envoie ce message DOIT ignorer tout SSH_MSG_UNIMPLEMENTED envoyé par le client en réponse. Si un serveur envoie à la fois ce message et un message SSH_MSG_USERAUTH_GSSAPI_ERROR, le message SSH_MSG_USERAUTH_GSSAPI_ERROR DOIT être envoyé en premier, pour permettre au client de mémoriser et/ou afficher l'état d'erreur avant de traiter le jeton d'erreur.

4. Authentification avec l'échange de clés GSS-API

Cette section décrit une méthode d'authentification d'utilisateur qui s'appuie sur le cadre décrit dans la [RFC4252]. Cette méthode effectue l'authentification d'utilisateur en faisant usage d'un contexte GSS-API existant établi durant l'échange de clés.

Le nom de la méthode d'authentification pour ce protocole est "gssapi-keyex".

Cette méthode ne peut être utilisée que si l'échange de clés initial a été effectué en utilisant un échange de clés fondé sur la méthode GSS-API définie conformément à la Section 2. Le contexte GSS-API utilisé avec cette méthode est toujours celui qui a été établi durant un échange de clés initial fondé sur GSS-API. Tout contexte établi durant l'échange de clés pour les besoins du changement de clés NE DOIT PAS être utilisé avec cette méthode.

Le serveur DEVRAIT inclure cette méthode d'authentification d'utilisateur dans la liste des méthodes qui peuvent continuer (dans un SSH_MSG_USERAUTH_FAILURE) si l'échange de clés initial a été effectué en utilisant un échange de clés fondé sur la méthode GSS-API et fournit des informations qui sont utiles au serveur sur l'identité de l'utilisateur. Il NE DOIT PAS inclure cette méthode si l'échange de clés initial n'a pas été effectué en utilisant un échange de clés fondé sur la méthode GSS-API définie conformément à la Section 2.

Le client DEVRAIT tenter d'utiliser cette méthode si elle est annoncée par le serveur, si l'échange de clés initial a été

effectué en utilisant un échange de clés fondé sur la méthode GSS-API, et si cette méthode n'a pas été déjà essayée. Le client NE DEVRAIT PAS essayer cette méthode plus d'une fois par session. Il NE DOIT PAS essayer cette méthode si l'échange de clés initial n'a pas été effectué en utilisant un échange de clés fondé sur la méthode GSS-API définie conformément à la Section 2.

Si un serveur reçoit une demande pour cette méthode quand un échange de clés initial n'a pas été effectué en utilisant un échange de clés fondé sur la méthode GSS-API définie en accord avec la Section 2, il DOIT retourner SSH_MSG_USERAUTH_FAILURE.

Cette méthode est définie comme un seul message :

byte SSH_MSG_USERAUTH_REQUEST
string nom d'utilisateur
string service
string "gssapi-keyex"
string MIC

Le contenu du champ MIC est obtenu en invoquant GSS_GetMIC sur ce qui suit, en utilisant le contexte GSS-API qui a été établi durant l'échange de clés initial :

string identifiant de session
byte SSH_MSG_USERAUTH_REQUEST
string nom d'utilisateur
string service
string "gssapi-keyex"

À réception de ce message quand l'échange de clés initial a été effectué en utilisant un échange de clés fondé sur la méthode GSS-API, le serveur utilise GSS_VerifyMIC() pour vérifier que le MIC reçu est valide. Si le MIC n'est pas valide, le champ Authentification d'utilisateur échoue, et le serveur DOIT retourner SSH_MSG_USERAUTH_FAILURE.

Si le MIC est valide et si le serveur est satisfait des accréditifs de l'utilisateur, il PEUT retourner soit SSH_MSG_USERAUTH_SUCCESS, soit SSH_MSG_USERAUTH_FAILURE avec le fanion de succès partiel établi, selon que des authentifications supplémentaires sont ou non nécessaires.

5. Algorithme de clé d'hôte Nul

L'algorithme de clé d'hôte "nul" n'a pas de matériel de clé d'hôte associé et ne fournit pas d'algorithme de signature ni de chiffrement. Donc, il peut être utilisé seulement avec les méthodes d'échange de clés qui n'exigent aucune opération de clé publique et n'exigent pas l'utilisation de matériel de clé publique d'hôte. Les méthodes d'échange de clés décrites à la Section 2 sont des exemples de telles méthodes.

Cet algorithme est utilisé quand, selon la configuration, l'hôte n'utilise pas ou ne souhaite pas utiliser une clé publique. Par exemple, il peut être utilisé quand l'administrateur a décidé selon sa politique d'exiger que tous les échanges de clés soient authentifiés en utilisant Kerberos [RFC4120], et donc la seule méthode d'échange de clés permise est l'échange Diffie-Hellman authentifié par GSS-API décrit ci-dessus, avec Kerberos V5 comme mécanisme GSS-API sous-jacent. Dans une telle configuration, la mise en œuvre de serveur prend en charge l'algorithme de clé "ssh-dss" (comme exigé par la [RFC4253]) mais il pourrait être interdit de l'utiliser par la configuration. Dans cette situation, le serveur a besoin d'un algorithme d'échange de clés pour les annonces ; l'algorithme "nul" sert à cela.

Noter que l'utilisation de l'algorithme "nul" de cette façon signifie que le serveur ne sera pas capable d'interopérer avec les clients qui ne supportent pas cet algorithme. Ce n'est pas un problème significatif, car dans la configuration décrite, il sera aussi être incapable d'interopérer avec les mises en œuvre qui ne supportent pas l'échange de clés authentifié par GSS-API et Kerberos.

Toute mise en œuvre qui prend en charge au moins une méthode d'échange de clés qui se conforme à la Section 2 DOIT aussi prendre en charge l'algorithme de clé d'hôte "nul". Les serveurs NE DOIVENT PAS annoncer l'algorithme de clé d'hôte "nul" sauf si il est le seul algorithme annoncé.

6. Résumé des numéros de message

Les numéros de message suivants ont été définis pour être utilisés avec l'échange de clés fondé sur les méthodes GSS-API :

```
#define SSH_MSG_KEXGSS_INIT           30
#define SSH_MSG_KEXGSS_CONTINUE       31
#define SSH_MSG_KEXGSS_COMPLETE       32
#define SSH_MSG_KEXGSS_HOSTKEY        33
#define SSH_MSG_KEXGSS_ERROR          34
#define SSH_MSG_KEXGSS_GROUPREQ       40
#define SSH_MSG_KEXGSS_GROUP          41
```

Les numéros 30-49 sont spécifiques de l'échange de clés et pourront être redéfinis par d'autres méthodes kex.

Les numéros de message suivants ont été définis pour être utilisés avec la méthode d'authentification d'utilisateur "gssapi-with-mic" :

```
#define SSH_MSG_USERAUTH_GSSAPI_RESPONSE                60
#define SSH_MSG_USERAUTH_GSSAPI_TOKEN                   61
#define SSH_MSG_USERAUTH_GSSAPI_EXCHANGE_COMPLETE       63
#define SSH_MSG_USERAUTH_GSSAPI_ERROR                   64
#define SSH_MSG_USERAUTH_GSSAPI_ERRTOK                  65
#define SSH_MSG_USERAUTH_GSSAPI_MIC                     66
```

Les numéros 60-79 sont spécifiques de l'authentification d'utilisateur et pourront être redéfinis par d'autres méthodes d'authentification d'utilisateur. Noter que dans la méthode décrite dans ce document, le numéro de message 62 est inutilisé.

7. Considérations sur GSS-API

7.1 Conventions de désignation

Afin d'établir un contexte de sécurité GSS-API, le client SSH a besoin de déterminer le targ_name *(nom de cible)* approprié à utiliser pour identifier le serveur lors de l'invocation de GSS_Init_sec_context(). À cette fin, le mécanisme de forme de dénomination indépendante de GSS-API pour les services fondés sur l'hôte est utilisé, comme décrit au paragraphe 4.1 de la [RFC2743].

En particulier, le targ_name à passer à GSS_Init_sec_context() est obtenu en invoquant GSS_Import_name() avec un input_name_type *(type de nom d'entrée)* de GSS_C_NT_HOSTBASED_SERVICE, et une input_name_string *(chaîne de nom d'entrée)* consistant en la chaîne "host@" enchaînée avec le nom d'hôte du serveur SSH.

Parce que le mécanisme GSS-API utilise le targ_name pour authentifier l'identité du serveur, il est important qu'il soit déterminé de façon sûre. Une façon courante de le faire est de construire le targ_name à partir du nom d'hôte comme entré par l'utilisateur ; malheureusement, parce que certains mécanismes GSS-API ne canonisent pas les noms d'hôtes, il est probable que cette technique va échouer si l'utilisateur n'a pas tapé un nom d'hôte pleinement qualifié, en forme canonique. Donc, les mises en œuvre peuvent souhaiter utiliser d'autres méthodes, mais devraient veiller à s'assurer qu'elles sont sûres. Par exemple, on ne devrait pas s'appuyer sur un enregistrement DNS non protégé pour transposer un alias d'hôte en nom principal d'un serveur, ou une adresse IP en un nom d'hôte, car un attaquant peut modifier la transposition et se faire passer pour le serveur.

Les mises en œuvre de mécanismes conformes au présent document NE DOIVENT PAS utiliser les résultats d'interrogations non sûres du DNS pour construire le nom cible. Les clients PEUVENT utiliser une transposition fournie par la configuration locale ou utiliser d'autres moyens sûrs pour déterminer le targ_name à utiliser. Si un système client est incapable de déterminer en toute sécurité quel targ_name utiliser, il NE DEVRAIT PAS utiliser ce mécanisme.

7.2 Liens de canaux

Le présent document recommande que les liens de canaux NE DEVRAIENT PAS être spécifiés dans les appels durant l'établissement de contexte. Le présent document ne spécifie aucune données standard à utiliser comme liens de canal , et l'utilisation des adresses réseau comme liens de canal peut casser SSH dans des environnements où il est très utile.

7.3 SPNEGO

L'utilisation du mécanisme de négociation GSS-API simple et protégé (SPNEGO) [RFC4178] en conjonction avec les méthodes d'authentification et d'échange de clés décrites dans le présent document est à la fois inutile et indésirable. Par suite, les mécanismes qui se conforment au présent document NE DOIVENT PAS utiliser SPNEGO comme mécanisme GSS-API sous-jacent.

Comme SSH effectue sa propre négociation de méthodes d'authentification et d'échange de clés, la capacité de négociation de SPNEGO seule n'apporte aucun avantage supplémentaire. En fait, comme décrit ci-dessous, il peut résulter en l'utilisation d'une méthode plus faible que désiré.

Normalement, SPNEGO apporte l'avantage supplémentaire de protéger la négociation de mécanisme GSS-API. Il fait cela en ayant le serveur qui calcule un MIC de la liste des mécanismes proposés par le client, et ensuite en vérifiant cette valeur chez le client. Dans le cas de l'échange de clés, cette protection n'est pas nécessaire parce que les méthodes d'échange de clés décrites ici effectuent déjà une opération équivalente, à savoir qu'elles génèrent un MIC du hachage de l'échange SSH, qui est un hachage de plusieurs éléments incluant les listes des mécanismes d'échange de clés pris en charge par les deux côtés. Dans le cas de l'authentification d'utilisateur, la protection n'est pas nécessaire parce que la négociation se fait sur un canal sûr, et que l'identité de l'hôte a déjà été prouvée à l'utilisateur.

L'utilisation de SPNEGO combinée avec les mécanismes GSS-API utilisés sans SPNEGO peut conduire à des problèmes d'interopérabilité. Par exemple, un client qui prend en charge l'échange de clés utilisant le mécanisme GSS-API Kerberos V5 [RFC4121] seulement en dessous de SPNEGO ne va pas interopérer avec un serveur qui prend en charge l'échange de clés en utilisant seulement directement le mécanisme GSS-API Kerberos V5. Il en résulte que permettre que les mécanismes GSS-API soient utilisés à la fois avec et sans SPNEGO est indésirable.

Si la politique d'un client est de d’abord préférer l'échange de clés fondé sur la méthode GSS-API X, puis la méthode non GSS-API Y, puis la méthode fondée sur GSS-API Z, et si un serveur prend en charge les mécanismes Y et Z mais pas X, une tentative d'utiliser SPNEGO pour négocier un mécanisme GSS-API peut résulter en l'utilisation de la méthode Z alors que la méthode Y aurait été préférable. Par suite, l'utilisation de SPNEGO pourrait résulter en la subversion de l'algorithme de négociation pour les méthodes d'échange de clés comme décrit au paragraphe 7.1 de la [RFC4253] et/ou de l'algorithme de négociation pour les méthodes d'authentification d'utilisateur comme décrit dans la [RFC4252].

8. Considérations relatives à l'IANA

Conformément à la Section 8 de la [RFC4251] et au paragraphe 4.6 de la [RFC4250], le présent document fait les enregistrements suivants :

La famille de noms de méthode d'échange de clés SSH commençant par "gss-group1-sha1-" et ne contenant pas le signe arobase (@), pour désigner les méthodes d'échange de clés définies au paragraphe 2.3.

La famille de noms de méthode d'échange de clés SSH commençant par "gss-group14-sha1-" et ne contenant pas le signe arobase (@), pour désigner les méthodes d'échange de clés définies au paragraphe 2.4.

La famille de noms de méthode d'échange de clés SSH commençant par "gss-gex-sha1-" et ne contenant pas le signe arobase (@), pour désigner les méthodes d'échange de clés définies au paragraphe 2.5.

Tous les autres noms de méthode d'échange de clés SSH commençant par "gss-" et ne contenant pas le signe arobase (@), sont réservés pour de futures méthodes d'échange de clés définies en conformité avec le présent document, comme noté au paragraphe 2.6.

Le nom d'algorithme de clé publique SSH "nul", pour désigner l'algorithme de clé d'hôte NUL défini à la Section 5.

Le nom de méthode d'authentification d'utilisateur SSH "gssapi-with-mic", pour désigner la méthode d'authentification d'utilisateur GSS-API définie à la Section 3.

Le nom de méthode d'authentification d'utilisateur SSH "gssapi-keyex", pour désigner la méthode d'authentification d'utilisateur GSS-API définie à la Section 4.

Le nom de méthode d'authentification d'utilisateur SSH "gssapi" est à réserver, afin d'éviter des conflits avec les mises en œuvre qui prennent en charge une version antérieure de la présente spécification.

Le nom de méthode d'authentification d'utilisateur SSH "external-keyx" est à réserver, afin d'éviter des conflits avec les mises en œuvre qui prennent en charge une version antérieure de la présente spécification.

Le présent document ne crée pas de nouveau registre.

9. Considérations sur la sécurité

Le présent document décrit des protocoles d'authentification et d'échange de clés. À ce titre, les considérations sur la sécurité sont discutées tout au long du document.

Ce protocole dépend du protocole SSH lui-même, de la GSS-API, de tous les mécanismes GSS-API sous-jacents qui sont utilisés, et de tous les protocoles sur lesquels de tels mécanismes pourraient dépendre. Chacun de ces composants joue un rôle dans la sécurité de la connexion résultante, et chacun va avoir ses propres considérations de sécurité.

La méthode d'échange de clés décrite à la Section 2 dépend du mécanisme GSS-API sous-jacent pour fournir à la fois l'authentification mutuelle et les services d'intégrité par message. Si l'une ou l'autre de ces caractéristiques n'est pas prise en charge par un mécanisme GSS-API particulier, ou par une mise en œuvre particulière d'un mécanisme GSS-API, alors l'échange de clés n'est pas sûr et DOIT échouer.

Afin que la méthode d'authentification d'utilisateur "gssapi-keyx" soit utilisée, elle DOIT avoir accès aux informations d'authentification d'utilisateur obtenues comme effet collatéral de l'échange de clés. Si cette information n'est pas disponible, l'authentification DOIT échouer.

Révéler des informations sur la raison d'un échec d'authentification peut être considéré par certains sites comme un risque inacceptable pour la sécurité dans un environnement de production. Cependant, la disponibilité de cette information peut être précieuse à des fins de débogage. Donc, il est RECOMMANDÉ que les mises en œuvre fournissent un moyen pour contrôler, en fonction de leur politique, l'envoi des messages SSH_MSG_USERAUTH_GSSAPI_ERROR, SSH_MSG_USERAUTH_GSSAPI_ERRTOK, et SSH_MSG_KEXGSS_ERROR, et SSH_MSG_KEXGSS_CONTINUE contenant un jeton d'erreur GSS-API.

10. Remerciements

Les auteurs remercient les personnes suivantes de leur précieuse assistance et de leurs contributions au présent document : Sam Hartman, Love Hornquist-Astrand, Joel N. Weber II, Simon Wilkinson, Nicolas Williams.

Beaucoup du texte de description de l'échange de groupe DH a été emprunté à la [RFC4419], de Markus Friedl, Niels Provos, et William A. Simpson.

11. Références

11.1 Références normatives

[ASN1] Recommandation UIT-T X.690, "Règles de codage ASN.1 : spécification des règles de codage de base (BER) des règles de codage canoniques (CER) et des règles de codage distinctives (DER)", (1997), aussi norme ISO/CEI 8825-1:1998, novembre 1998.

[RFC1321] R. Rivest, "Algorithme de résumé de message MD5", avril 1992. *(Information)*

[RFC2045] N. Freed et N. Borenstein, "Extensions de messagerie Internet multi-objets (MIME) Partie 1 : Format des corps de message Internet", novembre 1996. *(D. S., MàJ par* 2184, 2231, 5335.)

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997. *(MàJ par* RFC8174)

[RFC2743] J. Linn, "Interface générique de programme d'application de service de sécurité, version 2, mise à jour 1", janvier 2000. (*MàJ par* RFC5554)

[RFC3066] H. Alvestrand, "Étiquettes pour l'identification des langues", BCP 47, janvier 2001. *(Obsolète, voir la* RFC4646.)

[RFC3629] F. Yergeau, "UTF-8, un format de transformation de la norme ISO 10646", STD 63, novembre 2003.

[RFC4250] S. Lehtinen et C. Lonvick, éd., "Numéros alloués du protocole Secure Shell (SSH)", janvier 2006. *(P.S. ; MàJ par* RFC8268)

[RFC4251] T. Ylonen et C. Lonvick, "Architecture du protocole Secure Shell (SSH)", janvier 2006. *(P.S. ; MàJ par* RFC8308)

[RFC4252] T. Ylonen et C. Lonvick, éd., "Protocole d'authentification Secure Shell (SSH)", janvier 2006. *(P.S .; MàJ par* RFC8308, 8332)

[RFC4253] C. Lonvick, "Protocole de couche Transport Secure Shell (SSH)", janvier 2006. *(P.S., MàJ par* RFC6668, 8268, 8308, 8332, 8709)

[RFC4254] T. Ylonen et C. Lonvick, éd., "Protocole de connexion Secure Shell (SSH)", janvier 2006. *(P.S. ; MàJ par* RFC8308)

[RFC4419] M. Friedl et autres, "Échange de groupes Diffie-Hellman pour le protocole Secure Shell (SSH) de couche Transport", mars 2006. *(P.S. ; MàJ par* RFC8270)

11.2 Références pour information

[RFC4013] K. Zeilenga, "SASLprep : Profil Stringprep pour les noms d'utilisateur et mots de passe", février 2005.

[RFC4120] C. Neuman et autres, "Service Kerberos d'authentification de réseau (V5)", juillet 2005. (*MàJ par* RFC4537, 5021, 6649, 7751, 8062, 8129, 8429)

[RFC4121] L. Zhu et autres, "Version 2 du mécanisme d'interface de programme d'application de service de sécurité générique (GSS-API) de Kerberos version 5", juillet 2005. (*MàJ* RFC1964) (*MàJ par les* RFC6542, 6649, 8062)) *(P.S.)*

[RFC4178] L. Zhu et autres, "Mécanisme de négociation de l'interface de programme d'application de service générique de sécurité (GSS-API) simple et protégé", octobre 2005. (*Remplace* RFC2478) *(P.S.)*

Adresse des auteurs

Jeffrey Hutzelman
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213
US
téléphone : +1 412 268 7225
mél : jhutz+@cmu.edu

Joseph Salowey
Cisco Systems
2901 Third Avenue
Seattle, WA 98121
US
téléphone : +1 206 256 3380
mél : jsalowey@cisco.com

Joseph Galbraith
Van Dyke Technologies, Inc.
4848 Tramway Ridge Dr. NE
Suite 101
Albuquerque, NM 87111
US
mél : galb@vandyke.com

Von Welch
University of Chicago
Distributed Systems Laboratory
701 E. Washington
Urbana, IL 61801
US
mél : welch@mcs.anl.gov

Déclaration complète de droits de reproduction

Remerciement
Le financement de la fonction d'édition des RFC est fourni par l'activité de soutien administatif de l'IETF (IASA, *IETF Administrative Support Activity*).