Secrets d'authentification sous Windows

Aurélien Bordes

aurelien26@free.fr

Résumé Cet article se propose d'expliquer les mécanismes d'authentification sous Windows en abordant différents thèmes comme les sessions d'authentification, les *Authentication Packages*, SSPI, les fournisseurs SSP, ... Puis nous nous intéresserons à la problématique de la mise en cache des *credentials*.

La grande majorité des études ont été réalisées avec Windows XP service pack 2. Si les principes sont généraux à tous les Windows de la famille Windows NT, il peut exister quelques différences.

1 Introduction

L'authentification, généralement précédée d'une identification, est le mécanisme qui consiste à prouver une identité dont se réclame une « entité ». C'est une étape obligatoire avant l'autorisation qui consiste à autoriser l'entité à réaliser telle ou telle action.

Pour un utilisateur Windows, l'authentification se résume uniquement à la saisie d'un nom d'utilisateur et d'un mot de passe à l'ouverture de sa session. Il est très loin de se douter qu'en réalité il s'authentifie à chaque fois qu'il consulte son courrier avec Outlook ou ouvre ses fichiers sur le réseau. Mais le système l'authentifie automatiquement et il n'est généralement pas obligé de ressaisir son mot de passe.

Cela est possible car Windows dispose d'un mécanisme complet pour authentifier les utilisateurs et offrir un dispositif de mise en cache permettant le SSO (Single Sign-On ou authentification unique).

2 Rappels

Certains rappels se veulent volontairement rapides et simplistes. Le lecteur est invité à consulter les nombreux articles sur le sujet. Concernant tout ce qui se rapporte à LM et NTLM, ainsi qu'aux protocoles associés, l'article d'Eric Glass [1], sérieux et complet, fait référence sur le sujet.

2.1 Condensats

Le terme *credential*, dont la traduction française pourrait être « authentifiant », « crédence » ou « preuve d'authentification », désigne un ou des élément(s) secret(s) permettant à un utilisateur de s'authentifier. Cela peut être son mot de passe, ou plus souvent des condensats (*hashs*) calculés à partir de son mot de passe. En effet, les mots de passe ne sont pas stockés en clair, ils sont transformés par une fonction à sens unique afin de ne garder qu'un condensat (*hash*) non réversible. Deux types de condensat cohabitent sous Windows : les condensats LM et les condensats NTLM.

Condensat LM C'est la forme la plus ancienne. La taille du mot de passe est au maximum de 14 caractères et l'alphabet utilisable est limité. Le calcul d'un condensat LM à partir d'un mot de passe est le suivant :

- l'alphabet est simplifié (notamment les minuscules sont transformées en majuscules) ;
- un bourrage (*padding*) de zéro est ajouté en fin du mot de passe pour le compléter à 14 caractères ;
- le mot de passe est scindé en deux, ce qui donne deux parties de 7 caractères, soit 56 bits la taille d'une clé DES ;
- chacune des deux parties est utilisée comme clé pour chiffrer une constante (`KGS !@#$%`) avec l'algorithme DES ;
- les deux résultats sont concaténés pour donner le condensat LM.

Ce qui donne (`Password[n]` fait référence au n$^{\text{ème}}$ caractère, codé sur un octet, du mot de passe étendu à 14 caractères, et `|` est l'opération de concaténation) :

```
LM = DES(Password[0..6], KGS!@#$%) |
     DES(Password[7..13], KGS!@#$%)
```

Le condensat LM n'est maintenant plus considéré comme sûr. En effet, il est assez aisé de retrouver un mot de passe à partir de son condensat LM (sa forme après le passage par la simplification de l'alphabet pour être précis). C'est pourquoi, il est recommandé de ne plus conserver le condensat LM dans la base SAM des comptes locaux d'un système Windows (KB299656[1]).

Condensat NTLM Apparu avec Windows NT 4 Service Pack 3, le condensat NTLM corrige les faiblesses du condensat LM et offre une meilleure protection :

- la taille des mots de passe est augmentée à 255 caractères (certaines interfaces graphiques limitaient la taille à 14 caractères pour assurer la compatibilité avec le condensat LM) ;
- le codage Unicode UCS-2 est utilisé pour le mot de passe, c'est-à-dire que chaque caractère est codé sur deux octets.

Le condensat NTLM est calculé à partir du mot de passe via la fonction MD4.

```
NTLM = MD4(Password)
```

2.2 Protocoles de défi/réponse LM, NTLM, NTLMv2

À partir de leurs condensats, il faut pouvoir authentifier les utilisateurs via des échanges publics. Trois protocoles sont disponibles sous Windows, tous basés sur un mécanisme de défi/réponse : LM, NTLM et NTLMv2. Hormis LM qui se base sur le condensat LM, tous les autres protocoles utilisent le condensat NTLM. À noter l'existence d'un mode dénommé `NTLMv2 Session Security`, très largement méconnu [3].

Dans toutes les explications qui suivent, on se place dans le cas d'un client souhaitant s'authentifier auprès d'un serveur.

[1] `http://support.microsoft.com/kb/299656`

NTLMv2 Session Security `NTLMv2 Session Security` n'est pas réellement un algorithme de défi/réponse au même titre que LM ou NTLM. Il s'agit plutôt d'un nouveau mode de calcul utilisable pour générer la réponse NTLM. Il modifie également le champ dédié à la réponse LM (l'algorithme LM ne pouvant plus être utilisé).

À partir de Windows 2000 SP3, puis avec Windows XP ou 2003, ce mode est systématiquement négocié par défaut. Cependant, son utilisation n'est que facultative.

LM Soit `ServerChallenge` un défi (*challenge*) proposé par le serveur et `LM` le condensat LM du client. Le condensat (qui fait 2×64 soit 128 bits) est étendu à 168 bits par l'ajout de zéros, ce qui donne 3 clés DES ($3 \times 56 = 168$).

`LM[n]` fait référence au n$^{\text{ième}}$ octet du condensat. Si $n \geq 16, LM[n] = 0$, car les bits de 129 à 168 valent 0.

```
ReponseLM = DES(LM[0..6], ChallengeServeur) |
            DES(LM[7..13], ChallengeServeur) |
            DES(LM[14..20], ChallengeServeur)
```

Si le mode `NTLMv2 Session Security` est activé, la réponse LM n'est pas générée : elle est remplacée par le défi du client.

NTLM La génération de la réponse NTLM est très semblable à celle de la réponse LM. La seule différence, c'est l'utilisation du condensat NTLM (également étendu à 168 bits par l'ajout de zéros pour former 3 clés DES) en lieu et place du condensat LM.

Si $n \geq 16, NTLM[n] = 0$, car les bits de 129 à 168 valent 0.

```
ReponseNTLM = DES(NTLM[0..6], ChallengeServeur) |
              DES(NTLM[7..13], ChallengeServeur) |
              DES(NTLM[14..20], ChallengeServeur)
```

Si le mode `NTLMv2 Session Security` est activé, la génération de la réponse NTLM change car un défi proposé par le client est introduit. Ce n'est alors plus directement le défi du serveur qui est utilisé, mais un défi temporaire, calculé à partir des défis du serveur et du client, dont on ne garde que les 8 premiers octets (indiqués par `[0..7]`). La suite de la génération de la réponse NTLM reste inchangée. Quant à la réponse LM, elle contient le défi du client (complétée à 192 bits par l'ajout de zéros).

```
ChallengeTemp = MD5(ChallengeServeur | ChallengeClient)[0..7]
ReponseLM = ClientChallenge
ReponseNTLM = DES(NTLM[0..6], ChallengeTemp) |
              DES(NTLM[7..13], ChallengeTemp) |
              DES(NTLM[14..20], ChallengeTemp)
```

NTLMv2 Apparu avec Windows NT 4 Service Pack 4, le protocole NTLMv2 corrige la principale faiblesse des protocoles LM et NTLM, à savoir l'utilisation de DES pour le calcul de la réponse. Cet algorithme, qui date de 1977, n'est actuellement plus considéré comme sûr. Par exemple, la

recherche exhaustive sur les clés (2^{56} possibilités) peut se réaliser dans un intervalle très court (certains records étant en moins de 24 heures). Il devient alors possible de retrouver le condensat utilisé (LM ou NTLM) à partir de la capture d'un défi/réponse.

Le protocole NTLMv2 apporte deux modifications importantes :
- l'utilisation systématique dans les calculs d'un défi généré par le client ;
- l'utilisation de l'algorithme HMAC-MD5 pour le calcul des réponses.

Lorsque ce protocole est utilisé, les réponses LM et NTLM sont remplacées par des réponses LMv2 et NTLMv2. Les réponses LMv2 conservent la même taille que les réponses LM pour des besoins de compatibilités (192 bits). Les réponses NTLMv2 sont de tailles variables et incluent diverses informations supplémentaires, en particulier un champ d'horodatage.

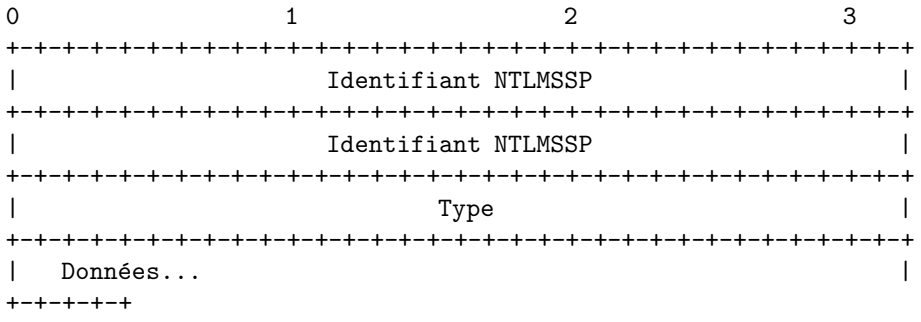
`NTLMv2Temp` correspond à un condensat temporaire calculé à partir du condensat NTLM, du nom de l'utilisateur mis en majuscule et du nom de la cible. `Bloc` représente une structure de données contenant diverses informations (dont le challenge du client) :

```
NTLMv2Temp = HMAC-MD5(NTLM, Majuscule(Nom utilisateur) | Nom cible)
ReponseLMv2 = HMAC-MD5(NTLMv2Temp, ChallengeServeur | ChallengeClient) |
              ChallengeClient
ReponseNTLMv2 = HMAC-MD5(NTLMv2Temp, ChallengeServeur | Bloc) |
                Bloc
```

Authentification avec le condensat Il est très important de remarquer que tous les protocoles d'authentification évoqués ci-dessus ne nécessitent pas le mot de passe, mais uniquement le condensat du mot de passe pour réussir le défi/réponse. Si un pirate réussit à se procurer le condensat LM ou NTLM d'un utilisateur, il pourra s'authentifier à distance avec le compte de cet utilisateur. Il existe d'ailleurs une version modifiée de Samba [4] permettant de s'authentifier non pas avec un mot de passe, mais uniquement avec un condensat LM ou NTLM.

2.3 NTLMSSP

Pour pouvoir mettre en œuvre les protocoles LM, NTLM et NTLMv2, il est nécessaire d'utiliser un protocole permettant de structurer les différents échanges. C'est le rôle du protocole NTLMSSP. L'en-tête d'un message NTLMSSP est de la forme :



Le champ `Identifiant NTLMSSP` est une constante permettant d'identifier un message NTLMSSP, dont la valeur est `NTLMSSP∅`. Quant au champ `Type`, il détermine le format des données qui suivent l'en-tête en indiquant le type de message NTLMSSP.

Trois types sont définis :
- **type 1 (`NTLMSSP_NEGOTIATE`)** : ce message est émis par le client pour démarrer un processus d'authentification. Optionnellement, des champs supplémentaires peuvent être présents comme le nom de sa machine, son domaine ou la version de son système d'exploitation ;
- **type 2 (`NTLMSSP_CHALLENGE`)** : ce message est émis par le serveur en réponse au message précédent. Sensiblement identique, il comporte toutefois un champ supplémentaire correspondant au défi du serveur (toujours sur 8 octets) ;
- **type 3 (`NTLMSSP_AUTH`)** : ce message est le plus important car il contient les réponses du client :
  - le champ LM, de taille fixe (24 octets, soit 192 bits) contient la réponse LM, LMv2 ou le défi du client lorsque le mode `NTLMv2 Session Security` est activé,
  - le champ NTLM contient la réponse NTLM ou NTLMv2.

Il faut également souligner la présence d'un champ `Flags` sur 32 bits dans les trois types de message qui permet d'indiquer la présence de champs optionnels ou le support de fonctionnalités. En particulier, le 24$^{\text{ème}}$ bit (soit `0x00800000`) indique si le mode `NTLMv2 Session Security` est supporté ou activé.

3 Fonctions de sécurité

Nous allons maintenant décrire la manière dont les composants Windows calculent les différents condensats et réponses LM ou NTLM. Il faut noter que les fonctions `SystemFunctionXXX` ne font pas partie de l'API Windows « officielle ». Leur utilisation nécessite donc de les charger manuellement avec `LoadLibrary` et `GetProcAddress`.

3.1 Condensats LM et NTLM

Pour calculer le condensat LM à partir d'un mot de passe, deux fonctions sont utilisées :
- `RtlUpcaseUnicodeStringToOemString`, définie dans la librairie `ntdll.dll` : cette fonction permet de convertir une chaîne de caractères Unicode en chaîne ASCII et de simplifier l'alphabet pour n'utiliser que les caractères dits « OEM » (utilisés par exemple pour les noms NetBios) ;
- `SystemFunction006`, définie dans la librairie `advapi32.dll` : cette fonction convertit une chaîne de caractères ASCII en condensat LM.

Le calcul d'un condensat NTLM est plus simple, car la simplification de l'alphabet n'est plus nécessaire. C'est uniquement la fonction `SystemFunction007` (définie dans `advapi32.dll`) qui est utilisée. Cette fonction ne fait que calculer un condensat MD4 à partir d'une chaîne de caractères Unicode (composée de `WCHAR`) ;

3.2 Réponses LM, NTLM et NTLMv2

Les réponses LM et NTLM sont calculées via la fonction `SystemFunction008` (définie dans `advapi32.dll`). Si le mode `NTLMv2 Session Security` est actif, il faut préalablement calculer un defit temporaire via la fonction MD5.

3.3 Utilisation des fonctions SystemFunction00*

Il est particulièrement intéressant de connaître le rôle des fonctions `SystemFunction006` et `SystemFunction007` car elles prennent en entrée une chaîne de caractères représentant un mot de passe en clair. Un programme, ou une librairie, qui utilise ces fonctions est probablement amené à manipuler des mots de passe, ce qui lui donne un intérêt certain. Dans le répertoire `System32`, on peut citer `msv1_0.dll`, `netapi32.dll`, `raschap.dll`, `samlib.dll`, `kerberos.dll`, ...

4 Formes d'authentification sous Windows

Deux formes d'authentification sont gérées par Windows :
- **interactives** : ces authentifications servent à ouvrir localement une session à la machine créant ainsi une session d'authentification (détaillée au paragraphe suivant). Plusieurs types d'authentification sont possibles, mais la plus connue est l'authentification interactive (d'où le nom) où les authentifiants sont saisis à l'aide d'une interface graphique (la GINA) et transmis au processus LSASS qui les valide à l'aide d'un *Authentication Package* (AP) ;
- **non-interactives** : ces authentifications interviennent après une authentification interactive. Elle permet à un client de s'authentifier auprès de serveurs distants en utilisant, éventuellement de manière transparente, les authentifiants saisis lors de l'authentification interactive. Ceci implique donc l'utilisation de mécanismes de mise en cache des *credentials* fournis ou calculés lors des authentifications interactives.

5 Sessions d'authentification

Le système Windows définit la notion de session d'authentification (*logon session*) qui représente un contexte d'authentification applicable à un processus ou un *thread*. Une telle session est représentée par une structure du type `SECURITY_LOGON_SESSION_DATA` qui contient, entre autres :
- l'identifiant unique de session, dénommé `logonID` (de type `LUID`[2]). Il est généré aléatoirement lors de la création de la session ou constant pour les sessions particulières ;
- le domaine et le nom de l'utilisateur ;
- le descripteur de sécurité (SID) de l'utilisateur ;
- le nom de l'*Authentication Packages* ayant validé et créé la session ;
- le type de session d'authentification (Interactive, Réseau, Service, ...).

C'est le processus LSASS qui est en charge de la gestion de la base contenant ces sessions. Plusieurs utilitaires existent pour consulter cette base [6]. Ils se basent généralement sur les fonctions `LsaEnumerateLogonSessions` et `LsaGetLogonSessionData`.

[2] un `LUID` est un identifiant unique (*Locally Unique IDentifier*), mais local à la machine par opposition à un `GUID` (*Globally Unique IDentifier*), globalement unique

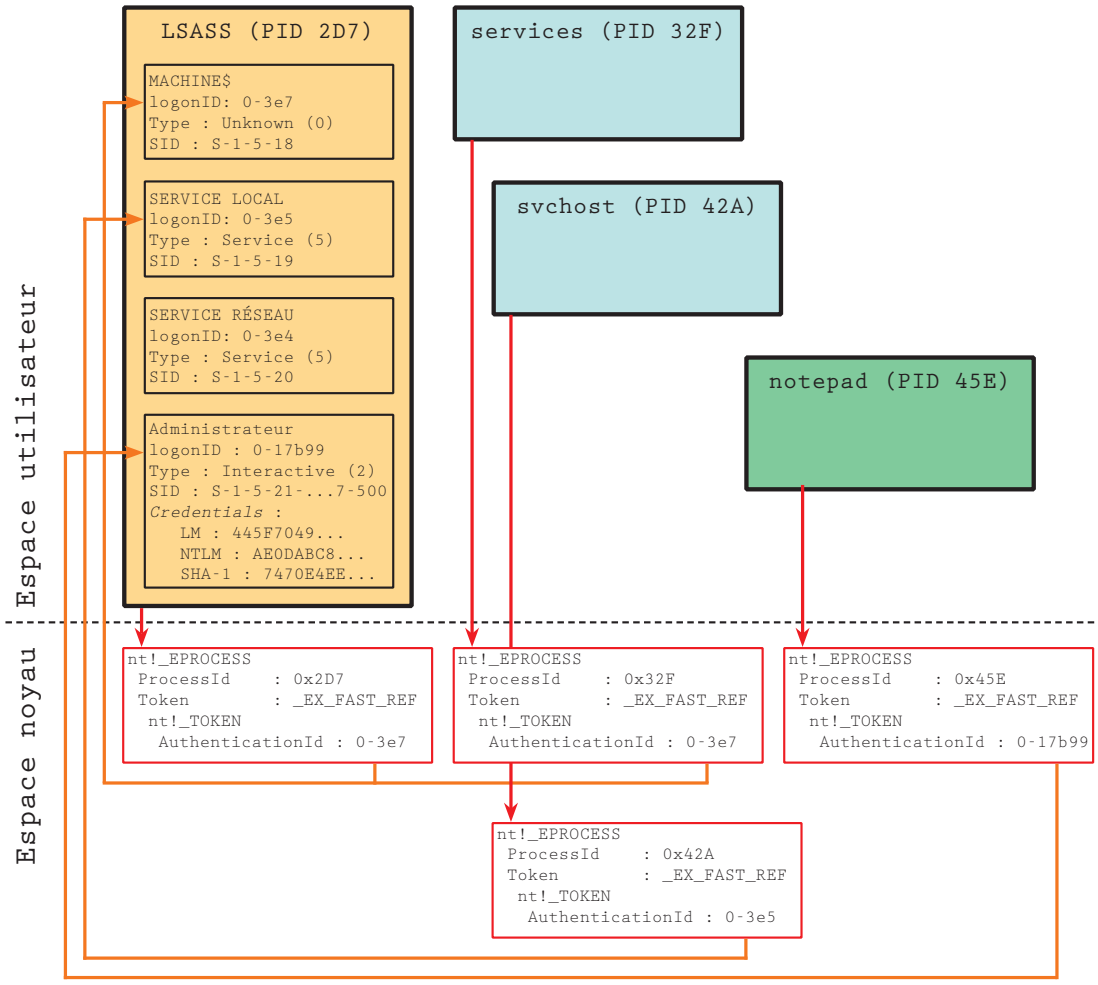


FIG. 1: Représentation des associations entre les processus, leur jeton d'accès et les sessions d'authentification

Pour chaque processus (ou *threads*), une session d'authentification est associée (voir schéma 1). Cette association est symbolisée par la présence dans la structure des jetons de sécurité (*security tokens* de type `nt!_TOKEN`) du champ `AuthenticationId`. Une session peut être associée à un ou plusieurs processus. L'exemple suivant permet avec WinDbg de consulter la valeur de ce champ pour un processus :

```
lkd> !process 0 1
**** NT ACTIVE PROCESS DUMP ****
...
PROCESS 89249988  SessionId: 0  Cid: 03c0    Peb: 7ffd5000  ParentCid: 01ec
```

```
    DirBase: 0b5805a0  ObjectTable: e11f4d20  HandleCount:  34.
    Image: notepad.exe
    VadRoot 89164b98 Vads 55 Clone 0 Private 142. Modified 0. Locked 0.
    DeviceMap e29e5358
    Token                             e1263980
...

lkd> dt -r1 nt!_TOKEN e1263980
...
   +0x010 TokenId          : _LUID
      +0x000 LowPart          : 0x112895
      +0x004 HighPart         : 0
   +0x018 AuthenticationId : _LUID
      +0x000 LowPart          : 0x17b99
      +0x004 HighPart         : 0
   +0x020 ParentTokenId    : _LUID
      +0x000 LowPart          : 0
      +0x004 HighPart         : 0
...
```

On commence par lister tous les processus puis, pour celui désiré, on affiche le contenu du jeton de sécurité primaire (*primary token*). Dans l'exemple, le processus `notepad.exe` est associé à la session d'authentification `0-17b99`.

6 Authentifications interactives

6.1 *Authentication Packages*

Lorsque qu'une « entité » souhaite se connecter localement à un système, la demande est adressée à LSASS qui va devoir valider l'authentification demandée. Cependant, cette validation n'est pas réalisée directement par LSASS, mais par des librairies dénommées *Authentication Packagess* (AP). Si la demande est acceptée, l'*Authentication Packages* génère un numéro unique de session, le `logonID`, et demande à LSASS de créer une session d'authentification associée à ce numéro via la fonction `CreateLogonSession`.

Lors du chargement d'un *Authentication Packages* par LSASS, celui-ci appelle la fonction `SpLsaModeInitialize` (obligatoirement exportée par un AP) en passant en paramètre une structure de type `SECPKG_FUNCTION_TABLE`. Cette structure contient un tableau de pointeurs de fonctions que l'AP doit remplir. Ces fonctions, implémentées par l'AP, seront appelées ultérieurement par LSASS. Les fonctions du type `LsaApLogonUserX` (`LsaApLogonUser`, `LsaApLogonUserEx` et `LsaApLogonUserEx2`) sont particulièrement importantes car elles sont responsables de la validation des demandes d'authentification transmises par LSASS.

La première fonction ainsi appelée par LSASS est `SpInitialize` laquelle finalise l'initialisation de l'AP. Une structure de type `LSA_SECPKG_FUNCTION_TABLE` est passée en paramètre et contient un tableau de fonctions de « soutien » implémentées par LSASS que pourra appeler l'AP. Parmi ces fonctions, on peut noter :

- `CreateLogonSession` : évoquée précédemment, cette fonction permet la création d'une *logon session* par LSASS ;
- `AddCredential` : cette fonction permet la mémorisation de *credentials*. En effet, LSASS offre la possibilité d'associer à une session d'authentification des *credentials* en les sauvant pour pouvoir les réutiliser par la suite. Lors de l'appel à la fonction, il faut spécifier le `logonID` d'une session, un numéro d'*Authentication Packages* ainsi qu'une chaîne de caractères servant de clé ;
- `GetCredentials` : complément de la fonction précédente, celle-ci permet de récupérer des *credentials* précédemment sauvés.

6.2 Validation par `msv1_0`

La grande majorité des validations d'authentification (surtout pour une machine autonome), est réalisée par l'*Authentifation Package* `msv1_0`, sollicité par LSASS via l'appel de la fonction `LsaApLogonUserEx2`, dont le prototype est :

```
NTSTATUS LsaApLogonUserEx2(
   PLSA_CLIENT_REQUEST [in] ClientRequest,
   SECURITY_LOGON_TYPE [in] LogonType,
   PVOID [in] ProtocolSubmitBuffer,
   PVOID [in] ClientBufferBase,
   ULONG [in] SubmitBufferSize,
   PVOID* [out] ProfileBuffer,
   PULONG [out] ProfileBufferSize,
   PLUID [out] LogonId,
   PNTSTATUS [out] SubStatus,
   PLSA_TOKEN_INFORMATION_TYPE [out] TokenInformationType,
   PVOID* [out] TokenInformation,
   PUNICODE_STRING* [out] AccountName,
   PUNICODE_STRING* [out] AuthenticatingAuthority,
   PUNICODE_STRING* [out] MachineName,
   PSECPKG_PRIMARY_CRED [out] PrimaryCredentials,
   PSECPKG_SUPPLEMENTAL_CRED_ARRAY* [out] SupplementalCredentials
);
```

Le paramètre `LogonType` indique le type d'authentification. Ce paramètre est fondamental, car il détermine la manière dont l'AP doit effectuer la validation.

Le plus simple est le type **Interactive** (`LogonType=2`). Il s'agit d'une authentification réalisée par un utilisateur, localement à la machine, dont la méthode la plus connue est l'ouverture de session physiquement sur la machine, via le clavier et l'écran. Mais le service `Terminal Server` ou la commande `runas` ouvrent également des sessions de type Interactive. Le processus de validation pour un utilisateur local est le suivant :

- la fonction `LsaApLogonUserEx2` est appelée. Le paramètre `ProtocolSubmitBuffer` représente une structure de données contenant l'identifiant et le mot de passe en clair de l'utilisateur, récupérés dans la majorité des cas par la GINA et transmis par WinLogon ;

- les condensats LM, NTLM du mot de passe sont calculés et stockés dans une structure qui contient également le nom de l'utilisateur et de la machine ;
- les condensats sont validés en les comparant avec ceux stockés dans la base SAM. S'ils sont différents, l'authentification s'arrête sur un échec ;
- une session d'authentification pour l'utilisateur est créée et la structure contenant le nom de l'utilisateur et les condensats est chiffrée (à partir de Windows XP), puis sauvegardée via l'appel de la fonction `AddCredential` offerte par LSASS (avec la clé `Primary`).

À partir de Windows XP, le condensat SHA-1 est également calculé et sauvé avec les condensats LM et NTLM. Il est utilisé en particulier par DPAPI [5].

L'autre type est le type **Service** (`LogonType=5`), utilisé pour créer les sessions d'authentification associées aux comptes de services. Il faut préalablement rappeler les différents comptes sous lesquels les services peuvent s'exécuter :

- **LocalSystem** : le processus tourne sous l'identité SYSTEM (SID `S-1-5-18`), c'est-à-dire avec beaucoup de privilèges. Le processus a la possibilité de s'authentifier sous l'identité de la machine car il est associé à la session d'authentification `0-3e7` (ce numéro de *logonID* est toujours identique). La création de cette session est détaillée dans le paragraphe 6.3 ;
- **NetworkService** : le processus tourne sous l'identité SERVICE RÉSEAU (SID `S-1-5-20`), mais avec peu de privilèges. Cependant, il conserve la possibilité de pouvoir s'authentifier sous l'identité de la machine et il est associé à la session `0-3e4` ;
- **LocalService** : le processus tourne sous l'identité SERVICE LOCAL (SID `S-1-5-19`), et dispose de peu de privilèges. Il n'a pas non plus la capacité de s'authentifier, car il est toujours associé à la session `0-3e5` à laquelle aucun *credential* n'est associé ;
- enfin, il est possible de spécifier explicitement un compte d'utilisateur et le mot de passe associé. Les privilèges et le contexte d'authentification sont alors ceux du compte utilisé.

C'est dans ce dernier cas que l'AP `msv1_0` va être sollicité. La procédure de validation est la même que pour celle de type Interactive mais avec une différence majeure : le mot de passe n'est pas fourni lors de l'appel de la fonction `LsaApLogonUserEx2`. L'AP va consulter les secrets de la LSA pour le trouver, en recherchant l'entrée correspondante au nom du service préfixé par `_SC_`. Par exemple, si le nom du service est `DemoSrv`, l'entrée `_SC_DemoSrv` contiendra le mot de passe à utiliser. La suite de la validation reste identique (y compris la sauvegarde des condensats).

L'exemple suivant montre un extrait des secrets de la LSA correspondant à l'entrée du mot de passe du compte associé au service `DemoSrv` :

```
_SC_DemoSrv
 50 00 61 00 73 00 73 00 31 00 32 00 20 00 21 00  P.a.s.s.1.2. .!.
 3A 00                                            :.
```

6.3 Authentification de la machine

Dans un domaine Windows 2000 ou 2003 utilisant l'Active Directory, les machines sont considérées comme des « acteurs » au même titre que les utilisateurs. Elles possèdent donc un identifiant (le nom de la machine suivi de $) ainsi qu'un mot de passe, connu par le système et l'Active Directory. Ils sont d'ailleurs renouvelés périodiquement, comme pour les mots de passe des comptes utilisateurs.

Cette capacité offerte aux machines de s'authentifier a permis de supprimer les accès anonymes à certains partages SMB ou services RPC.

Afin que des processus (essentiellement des services) soient capables de s'authentifier dans le contexte de la machine, il faut obligatoirement créer une session d'authentification qu'ils pourront utiliser. Cette session est créée pendant l'initialisation de LSASS sans passer par un processus d'authentification et ses caractéristiques sont :
- le nom de l'utilisateur qui correspond à celui de la machine, suffixé par `$` ;
- le SID qui est `S-1-5-18`, représentant le compte `SYSTEM` ;
- le *logonID* qui est toujours 0-3e7.

Le mot de passe utilisé pour générer les *credentials* associés à cette session va être lu dans les secrets de la LSA en consultant l'entrée `$MACHINE.ACC`. Si la machine n'est pas membre d'un domaine, l'entrée n'existant pas, aucun *credential* n'est associé. En revanche, si la machine est membre d'un domaine, l'entrée contient le mot de passe de la machine. Les condensats LM, NTLM et SHA-1 sont calculés puis sauvegardés via l'appel de la fonction `AddCredential` toujours associés à la clé `Primary`.

Il est important de préciser que le compte de la machine peut s'authentifier sur tous les systèmes membres du domaine (serveurs et postes clients). L'exemple suivant montre un extrait des secrets de la LSA correspondant à l'entrée du mot de passe de la machine :

```
$MACHINE.ACC
 D7 3F 94 CC 37 45 29 50 7A 16 BA B8 52 91 70 F6  .?..7E)Pz...R.p.
 83 02 AB 6F 64 7E 1B F4 A1 56 C3 15              ...od~...V..
```

7 Authentifications non-interactives

L'autre partie des composants d'authentification de Windows concerne la forme des authentifications dites non-interactives, c'est-à-dire permettant de réaliser une authentification à distance. Les informations contenues dans les sessions d'authentification précédemment créées peuvent éventuellement être utilisées.

7.1 *Security Support Provider*

Les authentifications non-interactives sont mises en œuvre via SSPI (*Security Support Provider Interface*) qui offre une interface de programmation unifiée aux applications. Les authentifications offertes par SSPI sont traitées par des fournisseurs (*providers*) appelés *Security Support Provider* (SSP). Les principaux fournisseurs de base distribués en standard par Microsoft sont :
- **Microsoft NTLM** (`msv1_0.dll`) qui prend en charge l'authentification via le protocole NTLMSSP et la mise en œuvre des trois algorithmes de défi/réponse LM, NTLM et NTLMv2. Il est fondamental pour ce package de disposer des condensats LM et NTLM pour générer correctement les réponses adéquates aux défis soumis ;
- **Microsoft Kerberos** (`kerberos.dll`) qui offre une authentification basée sur Kerberos v5 ;
- **Microsoft Negotiate** qui est un pseudo-fournisseur. Il permet de négocier l'utilisation du package NTLM ou Kerberos suivant des critères de sécurité et de compatibilité.

7.2 Mise en œuvre

Trois fonctions exportées par `secur32.lib` mettent en œuvre SSPI (voir figure 2) :
- `AcquireCredentialsHandle` : appelée côté client et serveur, cette fonction permet d'initialiser un fournisseur qui prendra en charge l'authentification ;
- `InitializeSecurityContext` : appelée côté client, cette fonction est utilisée pour initier et réaliser une authentification vers un serveur ;
- `AcceptSecurityContext` : appelée côté serveur, cette fonction permet de traiter et de répondre aux requêtes d'authentification émises par un client.

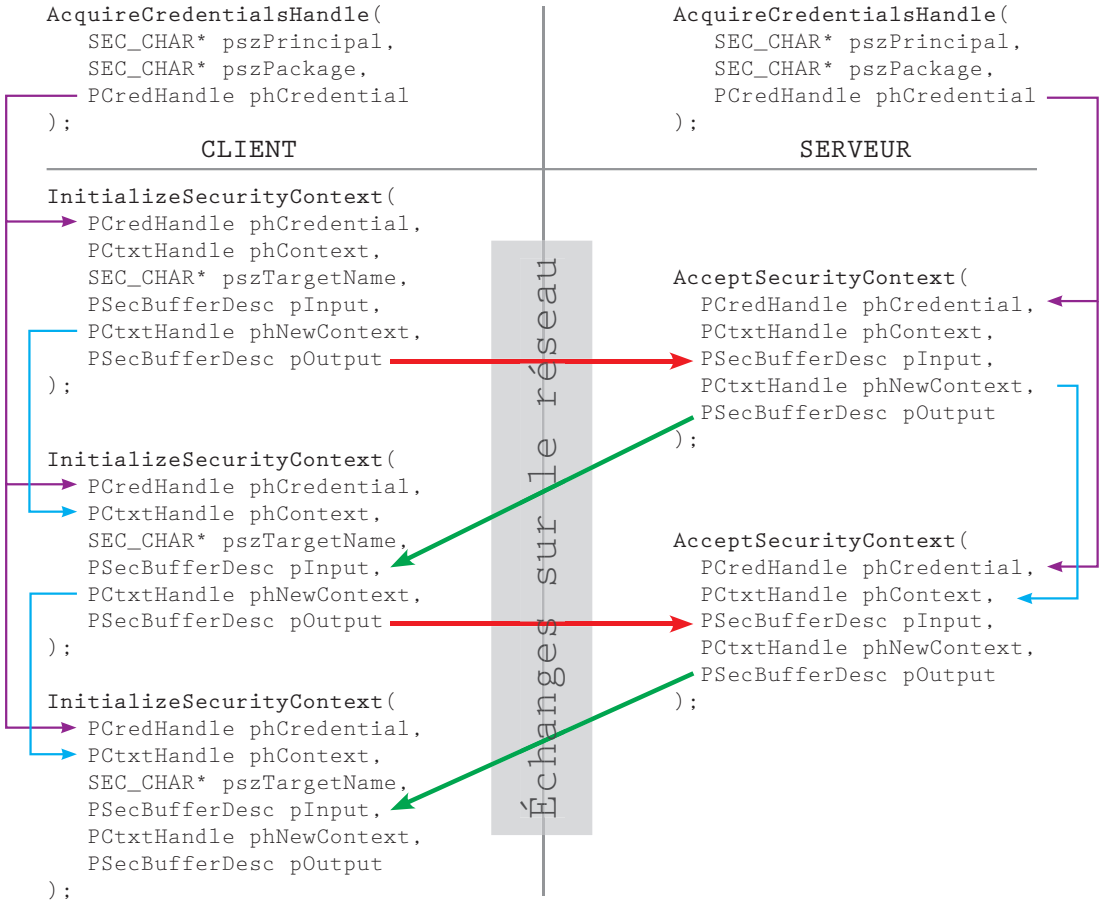


FIG. 2: Schéma des appels aux fonctions SSPI

Les fonctions `InitializeSecurityContext` et `AcceptSecurityContext` génèrent (paramètre `pOutput`) et acceptent (paramètre `pInput`) des blocs binaires qui doivent être transmis entre le client

et le serveur. Ces blocs doivent être encapsulés dans un autre protocole pour pouvoir être échangés sur le réseau. Par exemple, lorsque le mode « authentification intégrée Windows » est activé, Internet Explorer utilise les champs `Authorization` (client vers le serveur) et `WWW-Authenticate` (serveur vers le client) du protocole HTTP pour les transporter. L'utilisation avec le protocole SMB est détaillée dans le chapitre 9.

À la fin des échanges les fonctions rendent leur verdict (`SEC_E_OK` dans le cas d'une authentification réussie, `SEC_E_LOGON_DENIED` sinon).

Côté client, lors de l'appel à la fonction `AcquireCredentialsHandle`, il est possible de spécifier explicitement un nom d'utilisateur et un mot de passe à utiliser pour l'authentification via le paramètre d'entrée optionnel `pAuthData`. Celui-ci représente une structure du type `SEC_WINNT_AUTH_IDENTITY` dont la définition est :

```
typedef struct _SEC_WINNT_AUTH_IDENTITY {
  unsigned short __RPC_FAR* User;
  unsigned long UserLength;
  unsigned short __RPC_FAR* Domain;
  unsigned long DomainLength;
  unsigned short __RPC_FAR* Password;
  unsigned long PasswordLength;
  unsigned long Flags;
} SEC_WINNT_AUTH_IDENTITY, *PSEC_WINNT_AUTH_IDENTITY;
```

Si ce paramètre est renseigné, les authentifiants spécifiés (nom d'utilisateur et mot de passe) sont utilisés pour l'authentification. Dans le cas contraire (`pAuthData` à `NULL`), les *credentials* « par défaut » sont utilisés. Il n'est alors pas nécessaire de fournir des *credentials*, car ceux associés à la session d'authentification du programme appelant la fonction sont utilisés.

À noter le cas particulier d'un programme avec le privilège `SE_TCB_NAME` (agir en tant que partie du système d'exploitation) qui peut spécifier explicitement un identifiant de session d'authentification afin d'utiliser les *credentials* associés.

7.3 Récupération des *credentials* par le SSP

Il est important de bien comprendre la répartition des différents rôles. Une application cliente qui souhaite s'authentifier auprès d'un serveur en utilisant SSPI (par exemple via la mise en œuvre du SSP `msv1_0`) va recevoir un défi du serveur. Pour y répondre correctement (donc s'authentifier) elle doit calculer une réponse en utilisant les condensats LM ou NTLM. Le processus LSASS conserve pour chaque session d'authentification un jeu de *credentials* constitué des différents condensats, mais le processus ne peut évidement pas en disposer directement. Il doit donc transmettre à LSASS les défis reçus, qui calculera les réponses en utilisant les *credentials* associés à la session d'authentification du processus lui soumettant la requête. L'opération est identique lorsqu'un serveur reçoit des demandes d'authentification de la part d'un client : les messages reçus sont transmis à LSASS.

C'est par le mécanisme des LPC (*Local procedure call*) que la communication va s'opérer. Pour rappel, les LPC sont des RPC (*Remote Procedure Call*) limitées localement au système (en principe). Lorsque LSASS s'initialise, deux ports LPC sont créés :

- \`SeLsaCommandPort` est le port LPC permettant une communication exclusive entre le noyau et LSASS ;
- \`LsaAuthenticationPort` est le port LPC qui nous intéresse, car c'est celui que les applications ou le noyau utilisent pour adresser leurs demandes d'authentification.

La procédure de communication, par le biais de ce port, entre un processus et LSASS est la suivante :

1. dans le processus client, la fonction `AcquireCredentialsHandle` est appelée. Elle ouvre une connexion LPC vers LSASS puis demande l'acquisition des *credentials* de l'utilisateur ;
2. LSASS reçoit la demande et l'aiguilleur LPC dirige l'appel vers une fonction chargée d'obtenir des informations sur le processus appelant, en particulier son numéro de session d'authentification. Le mécanisme de changement d'identité (*impersonation*) est mis en œuvre et permet au *thread* traitant la demande d'agir temporairement avec un jeton de sécurité (*impersonation token*) identique à celui du processus client appelant la fonction LPC ;
3. La fonction consulte ensuite ce jeton de sécurité et deux informations sont récupérées :
   - le champ `AuthenticationId` qui contient un numéro de session d'authentification. Il s'agit donc de celui de la session du processus client,
   - la présence ou non d'identifiants de sécurité dans la liste des SIDs restreints (*restricted SIDs*) du jeton via un appel à la fonction `IsTokenRestricted`;
4. si la fonction `IsTokenRestricted` renvoie VRAI, l'appel s'arrête immédiatement, et la fonction `AcquireCredentialsHandle` renvoie l'erreur `0x8009030E` au client (`SEC_E_NO_CREDENTIALS` : No credentials are available in the security package). Il est donc impossible de continuer la procédure d'authentification. Dans le cas contraire, un *handle* de type `SecHandle` est retourné au client ;
5. le processus client poursuit son authentification par des appels à la fonction `InitializeSecurityContext` en spécifiant le handle récupéré précédemment. Si le processus LSASS est amené à manipuler des *credentials*, un appel à `GetCredentials` permettra de les récupérer. Le numéro de session passé en paramètre correspondra à celui récupéré lors de l'initialisation, c'est-à-dire celui de la session du client.

8 Outils de récupération

8.1 *Credentials* associés aux *logon sessions*

Principe Si des *credentials* constitués de condensats sont sauvés, il faut se prémunir contre le risque de divulgation de ceux-ci. Une technique de récupération consiste à lister toutes les sessions actives sur le système (voir chapitre 5), puis pour chacune d'entre elles, de récupérer les *credentials* associés via l'appel à la fonction `GetCredentials` offerte par LSASS. Cette fonction prend en paramètres :

- `LogonId` : l'identifiant unique de la session, récupéré via les fonctions d'énumération ;
- `AuthenticationPackage` : le numéro de l'*Authentication Packages*, récupéré également par les fonctions d'énumération.

La fonction retourne alors tous les *credentials* sauvés, ainsi que les clés spécifiées lors de la sauvegarde. La clé `Primary` indique les *credentials* sauvés par l'AP `msv1_0` lors de la création de la session. À partir de Windows XP, les structures contenant les *credentials* sont chiffrées ce qui nécessite de les déchiffrer préalablement.

Pour être autorisé à accéder aux *credentials*, il est nécessaire de s'injecter dans le processus LSASS (via un classique `OpenProcess`, `VirtualAllocEx`, `WriteProcessMemory`). La fonction `OpenProcess` nécessite le privilège `SeDebugPrivilege`, accordé uniquement aux Administrateurs. Les sessions sont alors énumérées via l'exécution (`CreateRemoteThread`) de la fonction injectée. Pour chaque session ouverte sur la machine, diverses informations sont retournées :

- le nom de l'utilisateur, son identifiant de sécurité (SID) et le numéro de la session d'authentification ;
- le nom de l'*Authentication Packages* ayant créé la session, ainsi que le type de session (interactive, réseau, service, ...) ;
- les *credentials* sauvés pour cette session, ainsi que les clés associées.

Exemple d'utilisation d'un outil de diagnostic :

```
Administrateur
 Type : Interactive (2)
 Session : 0, logonID : 0-17b99
 Authentication Package : NTLM
 SID : S-1-5-21-484888444-1244599845-468754557-500
  Primary credentials
  MACHINE\Administrateur
  LM : 445F70495C5F594018FCD526FB48A829
  NTLM : AE0DABC86921C8D0FB2F1EB234DC3AD7
  SHA-1 : 7470E4EE37E27BF990AFA1E57A083C2EE6247516

MACHINE$
 Type : Unknown (0)
 Session : 0, logonID : 0-3e7
 Authentication Package : NTLM
 SID : S-1-5-18

SERVICE LOCAL
 Type : Service (5)
 Session : 0, logonID : 0-3e5
 Authentication Package : Negotiate
 SID : S-1-5-19

SERVICE RÉSEAU
 Type : Service (5)
 Session : 0, logonID : 0-3e4
 Authentication Package : Negotiate
 SID : S-1-5-20
  Primary credentials
  DOMAINE\MACHINE$
  LM : AAD3B435B51404EEAAD3B435B51404EE
  NTLM : 31D6CFE0D16AE931B73C59D7E0C089C0
  SHA-1 : DA39A3EE5E6B4B0D3255BFEF95601890AFD80709
```

Dans cet exemple, le mot de passe de l'Administrateur est `sstic2007`. On constate bien la présence des sessions `0-3e4`, `0-3e5` et `0-3e7` utilisées par les comptes de service. La machine n'étant pas membre d'un domaine, il n'y a pas de *credential* associé à la session `0-3e7`. Quant à la session `0-3e4`, il s'agit des condensats correspondant à un mot de passe vide.

8.2 Sécurisation des mécanismes de défi/réponse

Le mécanisme des sessions d'authentification est plutôt bien conçu. Il permet lors de l'authentification interactive de mettre en cache des *credentials*, puis lors des authentifications non-interactives, de les utiliser, mais sans pouvoir les manipuler directement. S'il n'est pas possible à un utilisateur de récupérer ses *credentials* (donc potentiellement son mot de passe), il est cependant possible via SSPI de récupérer un échange de défi/réponse utilisant ses *credentials*.

Le principe pourrait consister à appeler dans un programme la fonction `AcquireCredentialsHandle` pour initialiser SSPI en demandant à utiliser les *credentials* par « défaut » de l'utilisateur (paramètre `pAuthData=NULL`) et le SSP `Microsoft NTLM` (paramètre `pszPackage="NTLM"` pour manipuler des messages NTLMSSP). Ensuite un défi (message NTLMSSP de type 2), généré de toutes pièces, est soumis à la fonction `InitializeSecurityContext`. On veillera dans ce défi à positionner le bit indiquant le support du mode `NTLMv2 Session Response` à 0 afin de ne pas le négocier (voir paragraphe 2.3). Un message NTLMSSP de type 3 est retourné (réponse) qui contient soit :

- une réponse LM et NTLM si le mode `NTLMv2 Session Security` n'est pas négocié ;
- un défi du client et une réponse NTLM modifiée si le mode `NTLMv2 Session Security` est négocié ;
- une réponse LMv2 et NTLMv2 si l'utilisation du protocole NTLMv2 est négocié.

La réponse du client dépend de la valeur du paramètre `LMCompatibilityLevel` (KB239869[3]). Dans tous les cas, un défi/réponse complet est récupéré. Il est alors possible de faire une attaque en force brute pour essayer de retrouver le condensat LM ou NTLM, puis le mot de passe.

Il est important de souligner que cette technique est possible pour tous les utilisateurs et ne requiert pas de droits particuliers. Comme on a vu dans le paragraphe 7.3 un tel programme ne fonctionne pas si son jeton d'accès contient des SIDs restreints. Pour exécuter un programme en restreignant ses droits et privilèges, il faut utiliser un logiciel spécialisé du type [2] ou [7]. Depuis Windows XP, l'explorateur Windows permet nativement cette fonctionnalité en utilisant le bouton droit de la souris, puis en sélectionnant l'option « Executer en tant que... » et « Protéger mon ordinateur et mes données des programmes non autorisés ».

`Auto-Auth.exe` Exemple avec un compte dont le mot de passe est `sstic2007` et la clé `LMCompatibilityLevel` ≤ 1 :

```
[NTLM Message Type] 2 (NTLMSSP_CHALLENGE)
[NTLM Challenge] 05deac1efc002a09

[NTLM Message Type] 3 (NTLMSSP_AUTH)
```

[3] http ://support.microsoft.com/kb/239869

```
[Lan Manager Response] 19eb24a7047091f2ce88574ec575c844ed5546ecc2599286
[NTLM Response] c163de084d2553ba55f40963b330b205798f91a645f0706c
```

Si la clé `LMCompatibilityLevel=2`, la réponse LM n'est plus générée et est identique à la réponse NTLM :

```
[NTLM Message Type] 2 (NTLMSSP_CHALLENGE)
[NTLM Challenge] 05deac1efc002a09

[NTLM Message Type] 3 (NTLMSSP_AUTH)
[Lan Manager Response] c163de084d2553ba55f40963b330b205798f91a645f0706c
[NTLM Response] c163de084d2553ba55f40963b330b205798f91a645f0706c
```

Si la clé `LMCompatibilityLevel` est ≥ 3, les réponses LMv2 et NTLMv2 sont systématiquement renvoyées :

```
[NTLM Message Type] 2 (NTLMSSP_CHALLENGE)
[NTLM Challenge] 05deac1efc002a09

[NTLM Message Type] 3 (NTLMSSP_AUTH)
[Lan Manager Response] ce61560b92616aa1f5f51d9a651bd0bd76e078c9956fa7bf
[NTLMv2 Response]
Domain name: MACHINE
User name: Administrateur
```

Parades Afin de se prémunir contre d'éventuels codes malveillants utilisant ce mécanisme, il faut appliquer ces trois règles :
- positionner la clé `LMCompatibilityLevel` à une valeur ≥ 3, l'idéal étant 5 ;
- choisir un mot de passe répondant à des critères sérieux en termes de taille et de complexité ;
- renouveler périodiquement son mot de passe.

9 Application au protocole CIFS/SMB

9.1 Rappel

SMB *(Server Message Block)* – ou CIFS *(Common Internet File System)* – est le protocole utilisé sous Windows pour le partage de fichiers. Il prend également en charge les canaux nommés via le partage IPC$, ce qui inclut les RPC sur canaux nommés (`ncacn_np`). Il est donc très largement utilisé.

Lors de la mise en place d'une connexion SMB (voir figure 3), les deux premiers échanges (messages `Negotiate Protocol Request`) permettent de négocier le dialecte SMB. Ensuite, des messages `Session Setup AndX Request` sont échangés afin de réaliser l'authentification. Le protocole SMB se base entièrement sur SSPI pour effectuer cette authentification entre le client et le serveur. Les blocs émis par les fonctions `InitializeSecurityContext` et `AcceptSecurityContext` sont échangés via le paramètre `Security Blob` des messages `Session Setup AndX Request`.

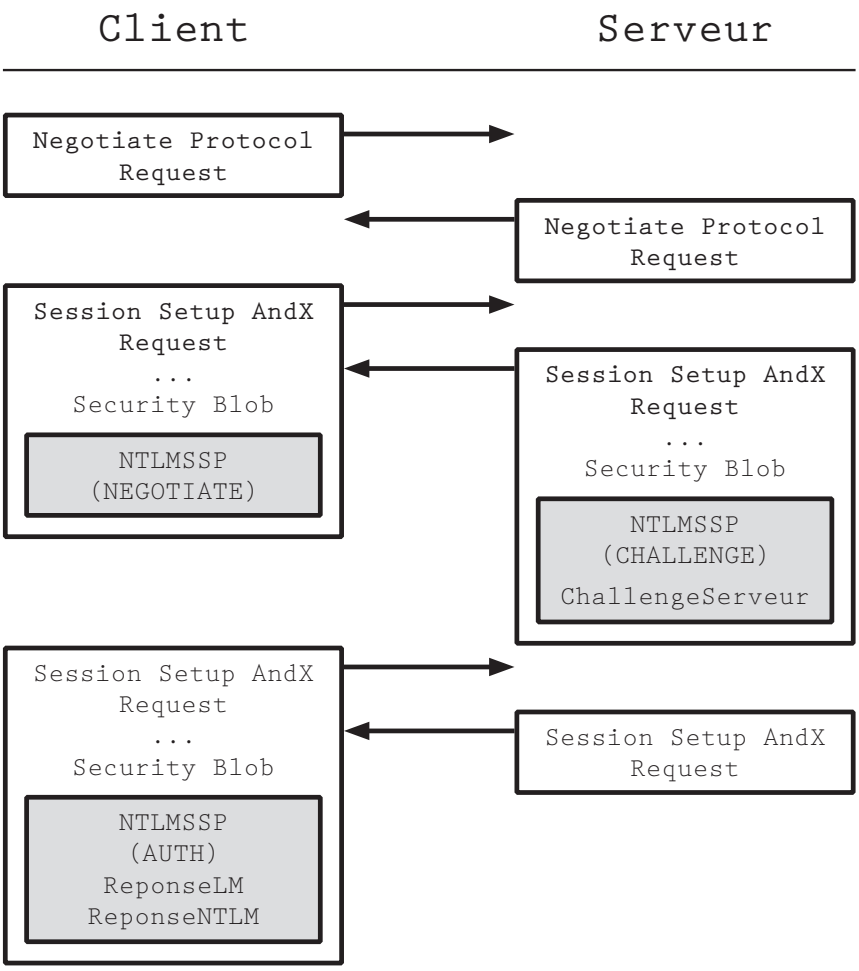


FIG. 3: Échanges SMB

Le fournisseur `Negotiate` est invoqué avec `cifs://<NomServeur>` comme nom de cible (paramètre `pszTargetName`). Celui-ci négocie l'utilisation du fournisseur `NTLM` ou `Kerberos` :
- si la machine est autonome, le fournisseur `NTLM` est systématiquement utilisé ;
- à partir de Windows 2000, si la machine est membre d'un domaine et que la cible est connue dans l'Active Directory, le fournisseur `Kerberos` est utilisé ;
- si la machine est membre d'un domaine mais que la cible n'est pas connue dans l'Active Directory (par exemple parce qu'elle est référencée par son adresse IP), le fournisseur `NTLM` est utilisé.

9.2 Authentification avec les authentifiants de la session

La commande suivante permet de se connecter au partage SMB appelé `C$` :

```
net use \\192.168.0.1\c$
```

Comme aucun nom d'utilisateur ni mot de passe n'est spécifié, il en sera de même lors de l'initialisation du fournisseur `Negotiate`. Ainsi l'authentification va être réalisée dans le contexte d'authentification de l'utilisateur initiant la commande, via l'utilisation des *credentials* par défaut.

9.3 Authentification en changeant d'utilisateur

Il est également possible de spécifier à la commande `net` un nom d'utilisateur et un mot de passe alternatif pour réaliser la connexion SMB :

```
net use \\192.168.0.1\c$ pass123 /user:Administrateur
```

Ces authentifiants sont transmis au fournisseur `Negotiate` qui peut ainsi réaliser l'authentification dans le contexte spécifié. À l'ajout du partage, c'est donc l'utilisateur qui fournit explicitement le mot de passe.

Mais il est intéressant de constater que le système est toujours capable de réussir une authentification LM ou NTLM tant que le partage n'est pas supprimé (côté client). Ceci indique qu'un mécanisme de cache conserve les condensats LM et NTLM ou plus simplement le mot de passe. Cependant, comme évoqué au paragraphe 7.2, le changement de contexte d'authentification ne peut se faire que lors de l'appel de la fonction `AcquireCredentialsHandle` en fournissant un nom d'utilisateur et un mot de passe en clair (structure `SEC_WINNT_AUTH_IDENTITY` via le paramètre `pAuthData`). C'est donc forcément le mot de passe en clair qui est sauvegardé dans un cache.

Ce cache se trouve dans le noyau. Le pilote (*driver*) SMB (`mrxsmb.sys`) maintient une base indiquant pour chaque machine distante les authentifiants à utiliser lorsque qu'une connexion à un partage SMB est réalisée. Pour chaque entrée de la base on trouve :
- le nom de la machine ;
- le nombre de partages actifs vers cette machine ;
- le numéro de la session d'authentification associée ;
- le nom de l'utilisateur et le domaine devant être utilisés pour l'authentification. Si cette information n'est pas présente, les *credentials* par défaut seront utilisés, c'est-à-dire ceux associés à la session ;
- le mot de passe en clair à utiliser si un nom d'utilisateur alternatif a été spécifié.

Un seul nom d'utilisateur peut être associé à une machine donnée. Si on tente d'utiliser un autre nom d'utilisateur, l'erreur 1219[4] est renvoyée.

9.4 Mesures de protection

La base gérée par le pilote `mrxsmb.sys` peut être consultée. Cette base étant située dans l'espace mémoire du noyau, il faut passer par l'écriture d'un pilote. Après le chargement de celui-ci, il localise la base et la parcourt pour retourner toutes les entrées. On retrouverait ainsi les informations énumérées au paragraphe précédent. Si pour une entrée donnée le nom d'utilisateur et le mot de passe ne sont pas présents, ce sont alors les authentifiants de la session associée qui sont utilisés.

Un tel utilitaire doit s'exécuter avec des privilèges d'Administrateur nécessaires pour l'installation du pilote. Une fois celui-ci installé, n'importe quel utilisateur peut consulter la base.

Illustration :

```
Base des sessions SMB :

 [\192.168.0.2] (2 références)
 logonID : 0-17b99
  Machine/Domaine : DOMTEST
  Utilisateur : (aucun)
  Mot de passe : (aucun)

 [\192.168.0.1] (1 référence)
 logonID : 0-17b99
  Machine/Domaine : DOMTEST
  Utilisateur : Administrateur
  Mot de passe : pass123
```

La première entrée indique une authentification réalisée avec les *credentials* associés à la session `0-17b99`.

Il est recommandé de supprimer les entrées inutiles de la base en se déconnectant des partages inutilisés. La commande `net use` permet de visualiser les partages actifs. La commande `net use /delete` permet de supprimer des partages. Lorsque tous les partages vers une machine donnée ont été supprimés, l'entrée est effacée de la base des caches.

```
C:\>net use
État         Local     Distant
-------------------------------------
OK           Z:        \\192.168.0.2\Partage
OK                     \\192.168.0.1\c$

C:\>net use /delete z:
```

[4] Le message d'erreur est : "Plusieurs connexions à un serveur ou à une ressource partagée par le même utilisateur, en utilisant plus d'un nom utilisateur, ne sont pas autorisées. Supprimez toutes les connexions précédentes au serveur ou à la ressource partagée et recommencez"

```
Z: a été supprimé.

C:\>net use /delete \\192.168.0.1\c$
\\192.168.0.1\c$ a été supprimé.

C:\>net use
La liste est vide.

C:\>dumpsmb.exe
Base des sessions SMB :

La base est vide.
```

10 Conclusion

L'authentification sous Windows est gérée par un dispositif particulièrement « sophistiqué » mais qui reste transparent pour l'utilisateur. Cet article a essayé d'éclaircir certains mécanismes, en particulier celui de la mise en cache des *credentials*.

Il faut garder à l'esprit qu'il est nécessaire de disposer de privilèges d'Administrateur pour manipuler directement les bases des caches et que l'application des bonnes pratiques usuelles est indispensable. Mais il est important connaître leur existence et le risque de divulgation suite à la compromission d'une machine.

Références

1. Glass, E. : The NTLM Authentication Protocol and Security Support Provider. `http://davenport.sourceforge.net/ntlm.html`
2. Howard, M. : Browsing the Web and Reading E-mail Safely as an Administrator. `http://msdn2.microsoft.com/en-us/library/ms972827.aspx`
3. Johansson, J. : The Most Misunderstood Windows Security Setting of All Time. `http://www.microsoft.com/technet/technetmag/issues/2006/08/SecurityWatch/default.aspx?loc=fr/`
4. JoMo-kun : Passing the Hash. `http://www.foofus.net/jmk/passhash.html`
5. NAI Labs, Network Associates : Windows Data Protection. `http://msdn2.microsoft.com/en-us/library/ms995355.aspx`
6. Russinovich, M. : LogonSessions. `http://www.microsoft.com/technet/sysinternals/SystemInformation/LogonSessions.mspx`
7. Russinovich, M. : PsExec. `http://www.microsoft.com/technet/sysinternals/utilities/psexec.mspx`