Dissection des RPC Microsoft

Nicolas Pouvesle[1] and Kostya Kortchinsky[2]

[1] npouvesle@tenablesecurity.com
[2] kostya.kortchinsky@eads.net

Résumé Cet article propose au lecteur une plongée au cœur des RPC Microsoft, ces appels de fonctions distants qui ont tant fait parler d'eux avec, entre autres, les vers Blaster, Sasser ou Zotob. La connaissance de leur fonctionnement détaillé est un atout majeur pour quiconque veut de nos jours sécuriser un réseau de machines sous Windows, ou ses propres applications client serveur communicant via RPC.

Nous présenterons les principes des RPC tels qu'ils sont implémentés dans Windows, en s'attardant sur certaines fonctionnalités intéressantes (que certains préféreront appeler vulnérabilités). Nous aborderons le langage de définition des interfaces MIDL, sa décompilation et son obscurcissement possible. Puis nous conclurons sur les renforcements possibles de la sécurité des mécanismes RPC, notamment les solutions retenues dans les derniers applicatifs Microsoft.

1 Introduction

L'acronyme RPC est maintenant ancré dans la mémoire des professionnels de la sécurité pour de tristes raisons, citons parmi celles-ci Blaster, Sasser, Zotob. Si l'organisation de ces appels distants de fonction (Remote Procedure Call) a longtemps été obscure, elle est maintenant mieux appréhendée. Les constats des diverses dissections menées à bien ne sont pas flatteurs...

En effet, l'utilisation du standard DCE RPC par Microsoft dans ses systèmes d'exploitation, aussi appelé MSRPC, concentre un éventail de vulnérabilités (ou fonctionnalités selon le point de vue) impressionnant, aussi bien dans la conception de l'architecture mise en œuvre que dans son implémentation. C'est sur ces fondations friables que se bâtissent des applicatifs, il convient donc de comprendre les mécanismes en jeu afin d'éviter à son édifice de s'effondrer à la moindre secousse.

Nous orienterons la suite de cet article dans une direction résolument technique, en insistant sur des points méconnus ou peu documentés de MSRPC, afin d'éviter des redites avec des documents existants. Après avoir introduit quelques concepts clés de RPC, nous aborderons le langage de description des interfaces, MIDL (Microsoft Interface Description Language) [MIDL], ainsi que les possibilités de rétro-conception offertes par celui-ci, puis nous nous attarderons sur les faiblesses structurelles de MSRPC, et ce, au travers d'exemples singuliers. Enfin, nous conclurons sur les efforts de sécurisation engagés par Microsoft avec les dernières versions de ses logiciels.

2 MSRPC, l'implémentation de DCE RPC par Microsoft

Le mécanisme de RPC permet d'appeler des fonctions sur un système distant de façon quasi transparente comme s'il s'agissait d'appels locaux. Parmi les deux principales implémentations existantes de RPC, Microsoft a implémenté dans ses produits DCE RPC en y ajoutant des fonctionnalités spécifiques à Windows.

2.1 Quelques concepts clés

Afin de comprendre le fonctionnement de MSRPC, il est nécessaire d'introduire deux notions importantes : celle d'interface (« interface » en Anglais), de protocole de transport, et de point final (« endpoint » en Anglais).

Un service RPC Windows peut être accessible via différents protocoles de transport, les plus fréquents étant :

- un pipe utilisé par le protocole réseau Windows SMB/CIFS (ncacn_np) ;
- le protocole réseau TCP (ncacn_ip_tcp) ;
- le protocole réseau UDP (ncacn_ip_udp) ;
- le protocole HTTP (ncacn_http) ;
- un pipe SMB/CIFS local (ncalrpc).

Lorsque le protocole de transport utilisé est SMB/CIFS le service RPC est en écoute sur les ports TCP 139 et 445. S'il s'agit de TCP ou UDP le port doit être précisé lors de la création du service. Un client peut connaître le port en interrogeant le service « portmapper » en écoute sur le port TCP 135.

L'ensemble des protocoles de transport utilisés par un service RPC est défini au sein d'un point final (« endpoint ») et a le format suivant :

```
endpoint("ncacn_ip_tcp:[2046]", "ncacn_np:[\ \
pipe\\rpc_service]");
```

L'exemple précédent représente un service RPC en écoute sur le port TCP 2046 et également disponible via le protocole réseau SMB/CIFS.

Un service RPC contient enfin une ou plusieurs interfaces servant à décrire toutes les fonctions utilisées ainsi que tous les points finaux disponibles. Une interface RPC est représentée par un identifiant (uuid (00000001-0002-0003-0004-000000000005)), un numéro de version (version (1.0)), des points finaux (optionnels). Le lecteur est invité à lire [WNSI] pour une liste complète des services RPC Windows.

L'ensemble des caractéristiques permettant de décrire un service MSRPC sont contenues dans un fichier texte (service.idl) utilisant le langage MIDL.

3 MIDL

MIDL est le langage qui permet de définir les interfaces entre des programmes client et serveur.

3.1 Présentation du langage MIDL

L'utilisation des appels de fonctions distantes RPC dans une application permet donc au développeur de ne plus avoir à se soucier de gérer la communication entre le client et le serveur. Cependant, il est toujours nécessaire de définir les interfaces utilisées. Une fois ces interfaces définies, le compilateur générera automatiquement le code permettant au client et au serveur de communiquer.

Les interfaces MSRPC sont, sous Windows, définies à l'aide du langage MIDL ("Microsoft Interface Definition Language" en anglais) qui est une extension du langage IDL utilisé pour représenter les interfaces DCE RPC.

Le langage MIDL est très proche du C et du C++ utilisés dans les fichiers entêtes. La principale différence réside dans la présence de mots clés précisant l'encodage utilisé.

Une interface de communication RPC est définie dans un fichier idl et a la forme suivante :

```
[
uuid(01234567-89ab-cdef-0123-456789abcdef),
version(1.0)
]

interface Example
{
typedef struct _info
 {
  long l;
  char c;
 } info;

long RunCommand (
 [in][string] char * cmd,
 [out] info * inf
 );
}
```

La première partie du fichier, comprise entre crochets, peut être qualifiée d'entête et contient les informations caractérisant l'interface :

```
[
uuid(01234567-89ab-cdef-0123-456789abcdef),
version(1.0)
]
```

On trouve, au minimum, deux éléments : l'identifiant unique (uuid) et la version. On peut également trouver dans l'entête des informations concernant les points finaux ("endpoints"), l'encodage (pointeur) et la manière d'identifier une session (handle).

Juste après l'entête on trouve le corps de l'interface :

```
interface Example
 {
  typedef struct _info
   {
    long l;
    char c;
   } info;
long Test (
 [in][string] char * tab,
 [out] info * inf
 );
}
```

Celui ci contient la définition de chaque fonction utilisée pour communiquer entre le client et le serveur mais également la définition de toutes les structures passées dans ces fonctions.

Si l'on regarde de plus près la fonction « Test », on s'aperçoit que chaque argument est précédé d'un mot clé ([in], [out] ou [in, out]). Ces mots clés servent à préciser dans quel sens les arguments doivent être transmis :

[in] : client vers serveur,
[out] : serveur vers client,
[in, out] : client vers serveur puis serveur vers client.

D'autres mots clés peuvent être présents :

```
long Test2 (
 [in][string] wchar_t * str,
 [in] long l,
 [in][size_is(l)] wchar_t * str2
 }
```

L'exemple précédent contient une grande partie des failles d'implémentation RPC actuellement connues. Il est donc nécessaire de bien le comprendre. L'argument « str » est une chaîne de caractères (string) unicode (wchar_t). Il n'est fait a aucun moment mention de la taille de la chaîne. La taille exacte de la chaîne sera passé au niveau de l'encodage et les fonctions RPC se chargeront d'allouer et désallouer les zones mémoires nécessaires. Un programme devra donc faire très attention à ne pas copier cet argument dans un buffer de taille fixe pour éviter d'éventuels débordements de tampons.

L'argument « str2 » est précédé du mot clé size_is qui sert à indiquer la taille réelle du buffer (différent de la taille réellement transmise). Il faut donc toujours vérifier cette taille avant toute manipulation.

En supposant que les arguments « str » et « str2 » ont une taille maximum, il est possible de modifier la définition IDL est ainsi de réduire les risques d'erreurs de programmation :

```
 long Test2 (
 [in][range_is(0,100)] long l1,
```

```
 [in][string][size_is(l1)] wchar_t * str,
 [in][range_is(0,10000)] long l2,
 [in][size_is(l2)] wchar_t * str2
 }
```

Le mot clé range_is forcera les fonctions RPC à vérifier que l'entier (l1 ou l2) est bien compris entre les valeurs spécifiées. Si ce n'est pas le cas une exception RPC aura lieu et la fonction ne sera jamais appelée.

Une fois l'interface MIDL définie, elle peut être transformée en code C à l'aide du compilateur midl.exe.

3.2 Encodage MIDL

Une analyse du code généré par le compilateur MIDL va permettre de comprendre comment il est possible d'extraire et de décompiler les interfaces RPC d'un binaire.

Le compilateur MIDL fourni par Microsoft offre de compiler l'interface IDL en stub interprété (fully interpreted stub), en stub semi-interprété (mixed stub ou interpreted stub) et en mode en ligne (inline stub). La différence entre ces différents modes réside dans le code généré mais également dans la vitesse de traitement des requêtes RPC (le mode en ligne étant le plus rapide).

Bien qu'il existe plusieurs modes de compilation, le code généré présente cependant des similitudes. On trouve, notamment, la structure décrivant l'interface du serveur :

```
static const RPC_SERVER_INTERFACE Example___RpcServerInterface =
    {
     sizeof(RPC_SERVER_INTERFACE),
 {{0x01234567,0x89ab,0xcdef,{\{}0x01,0x23,0x45,0x67,0x89,
        0xab,0xcd,0xef}},{1,0}},
 {{0x8A885D04,0x1CEB,0x11C9,{\{}0x9F,0xE8,0x08,0x00,0x2B,
        0x10,0x48,0x60}},{2,0}},
    &Example_v1_0_DispatchTable,
     0,
     0,
     0,
   &Example_ServerInfo,
     0
      };
```

On peut noter la présence de l'identifiant unique de l'interface « Example », suivi par l'identifiant représentant le protocole de transfert. Les deux autres entrées sont également très importantes. Juste après l'identifiant de transfert se situe un pointeur vers une structure appelée DispatchTable :

```
RPC_DISPATCH_TABLE Example_v1_0_DispatchTable =
 {
```

```
  1,
  Example_table
 };
```

Cette structure a comme premier élément le nombre de fonctions présentes dans l'interface RPC. Les autres éléments sont des pointeurs vers le code de chaque fonction RPC dans le cas d'un stub en ligne et un pointeur vers une structure de contrôle dans le cas d'un stub interprété.

L'autre élément de la structure RpcServerInterface, Example_ServerInfo, est un pointeur vers une autre structure contenant des informations sur l'interface RPC. Il est a noté que ce pointeur n'existe pas si l'on est dans le cas d'un stub en ligne. Cette structure a le format suivant :

```
static const MIDL_SERVER_INFO Example_ServerInfo =
 {
  &Example_StubDesc,
  Example_ServerRoutineTable,
  __MIDL_ProcFormatString.Format,
  Example_FormatStringOffsetTable,
  0,
  0,
  0,
  0
  };
```

Cette structure est essentiellement composée de pointeurs :

- Example_ServerRoutineTable est un pointeur sur une structure contenant des pointeurs vers le code de chaque fonction RPC ;
- _MIDL_ProcFormatString.Format est un pointeur vers une chaîne de format représentant chaque fonction (arguments, encodage...) ;
- Example_FormatStringOffsetTable est un pointeur sur une structure indiquant l'offset de chaque fonction RPC dans une chaîne de format servant à analyser l'encodage.

Le tout premier élément de la structure, StubDesc, pointe vers une nouvelle structure de contrôle :

```
static const MIDL_STUB_DESC Example_StubDesc =
 {
  (void __RPC_FAR *)&Example___RpcServerInterface,
  MIDL_user_allocate,
  MIDL_user_free,
  0,
  0,
  0,
  0,
  0,
  __MIDL_TypeFormatString.Format,
```

```
  1, /* -error bounds_check flag */
  0x20000, /* Ndr library version */
  0,
  0x50100a4, /* MIDL Version 5.1.164 */
  0,
  0,
  0, /* notify {\&} notify_flag routine table */
  1, /* Flags */
  0, /* Reserved3 */
  0, /* Reserved4 */
  0 /* Reserved5 */
 };
```

L'élément important de cette structure est __MIDL_TypeFormatString.Format. Il s'agit d'un pointeur sur une autre chaîne de format qui décrit les arguments plus complexes ainsi que les structures utilisées dans l'interface MIDL. A titre d'exemple, voici le code représentant l'argument str de l'interface « Test » :

```
0x11, 0x8, /* FC_RP [simple_pointer] */
/* 4 */
0x22, /* FC_C_CSTRING */
0x5c, /* FC_PAD */
```

On remarquera que cette chaîne de format est relativement simple à comprendre. Il s'agit d'un pointeur (0x11) sur un type de base (0x8) qui est une chaîne unicode (0x22).

Les structures précédentes ainsi que les chaînes de format étant présentes dans le binaire, il est tout à fait envisageable de pouvoir les extraire de ce binaire et de les décompiler afin d'obtenir le code MIDL originel de l'interface RPC.

3.3 Décompilation MIDL

On a vu précédemment qu'il était envisageable de décompiler les chaînes de format RPC afin de recréer le code MIDL. Il faut cependant pouvoir trouver ces chaînes au sein de l'éxécutable.

En regardant de plus près les différentes structures composant le code RPC, on s'aperçoit que, dans le cas d'un stub complètement interprété (le plus fréquent), il est possible de trouver les chaînes de format en suivant les pointeurs de la structure Example__RpcServerInterface. Or cette structure dispose d'un élément stable et facilement identifiable : l'identifiant du protocole de transfert.

La décompilation du code RPC d'un stub interprété et/ou complètement interprété est donc possible. Mais qu'en est il d'un stub en ligne ?

Il n'existe aucune façon fiable d'appliquer la même démarche que précédemment car le pointeur Example_ServerInfo n'existe pas s'il s'agit d'un stub inline. Cependant dans ce cas là, la structure DispatchTable est une liste de pointeur dans le code binaire du programme sur chaque fonction. A la différence d'un stub interprété où le code des arguments RPC sont analysé hors des fonctions, chaque

fonction de l'interface RPC d'un stub inline analyse ces arguments (code généré automatiquement) :

```
NdrServerInitializeNew(
_pRpcMessage,
&_StubMsg,
&Example_StubDesc);
[...]
NdrConvert( (PMIDL_STUB_MESSAGE) &_StubMsg,
            (PFORMAT_STRING)
&__MIDL_ProcFormatString.Format[0] );
```

Une fonction fait donc référence à Example_StubDesc qui contient les pointeurs nécessaires pour trouver une des chaînes de format et fait directement référence à l'autre chaîne de format.

Il est donc également possible d'extraire et de reconstruire le code MIDL dans le cas d'un stub en ligne. Il faut cependant être capable d'analyser le code binaire d'un programme.

Il existe actuellement plusieurs décompilateurs MIDL permettant de recréer le code MIDL :

- muddle [MUDDLE] : premier décompilateur MIDL apparu en 2000 et fait par Matt Chapman. Ce décompilateur, afin de trouver les différentes structures RPC utilise comme code fixe la version de la librairie Ndr. Cette méthode n'est cependant pas fiable et ne permet pas de gérer correctement les stubs en ligne. La détection des stubs en ligne est basée sur le fait que la chaîne de format se trouve à une position fixe par rapport à la structure découvert. Or ce n'est rarement le cas. muddle n'est plus maintenu et ne permet de recréer le code MIDL qu'avec du code compilé avec des vieilles versions de midl.exe.

- unmidl [UNMIDL] : décompilateur MIDL écrit en python en 2004-2005 par Dave Aitel. Ce décompilateur MIDL permet d'extraire les interface MIDL sans avoir à faire aucune analyse préalable sur le binaire. Il fonctionne sur toutes les plateformes permettant d'utiliser python (Unix, Windows, ...). Il présente cependant le même problème que muddle au niveau des stub en ligne.

- mIDA [MIDA] : plugin IDA écrit en 2005-2006 par Nicolas Pouvesle. Il s'agit d'un plugin pour le désassembleur IDA. Il est donc nécessaire de posséder une copie de ce logiciel (Windows). Il offre cependant un meilleur support de l'analyse des stubs en ligne en s'appuyant sur l'analyse du code binaire effectuée par le désassembleur.

Il est donc tout à fait possible d'extraire et de reconstruire le code MIDL d'un binaire en utilisant des outils existant ou en créant soit même de nouveau programme.

Cependant existe-t-il des méthodes permettant d'empêcher cette décompilation.

3.4 Obscurcissement IDL

Les chaînes de format générées par le compilateur MIDL de Microsoft étant nécessaires au fonctionnement d'une application RPC il n'est pas possible de les

supprimer et donc d'empêcher quelqu'un de recréer le code. Cependant il existe quelques techniques permettant de rendre cette tache beaucoup plus ardue.

On peut dans un premier temps s'attaquer à l'obscurcissement des données caractéristiques de l'encodage RPC. Il est par exemple possible de supprimer l'identifiant du protocole de transfert ainsi que le numéro de version de la librairie Ndr. Ces champs devront cependant être remplies avant d'appeler les fonctions RPC. De cette façon les décompilateurs n'ont plus de point de repère fixe permettant de trouver ces structures.

Il est cependant possible de rechercher les appels aux fonctions d'initialisation RPC ce qui permet de suspecter la présence d'un telle structure et ainsi d'en obtenir l'emplacement dans le binaire.

Un autre technique, plus subtile, consiste à modifier le contenu des deux chaînes de format. Le code généré par les décompilateurs RPC n'aura alors plus rien à voir avec la réalité. Là encore il est nécessaire de restaurer ces chaînes avant d'appeler les fonction d'initialisation RPC (RpcServerRegisterIf,...). Cette deuxième méthode est beaucoup plus intéressante que la première si elle est correctement mise en oeuvre. En effet, en remplaçant les chaînes de format avec d'autres chaînes de format RPC cette obscurcissement peut facilement passer inaperçu.

Enfin il existe une dernière technique d'obscurcissement MIDL qui ne concerne que certains décompilateurs et n'est valable que dans le cas d'un mode en ligne. Supposons que le fichier IDL ressemble à cela :

```
/* opcode 0x00 */
long function1 (
[in] long l
);
/* opcode 0x01 */
long function2 (
[in] char c,
[out] long * l
);
```

Il suffit alors de créer des fonctions fantômes qui ne servent à rien mais qui ont exactement les mêmes définitions. Le fichier IDL deviendrait par exemple :

```
/* opcode 0x00 */
long function_ph1 (
[in] char c,
[out] long * l
);
/* opcode 0x01 */
long function_ph2 (
[in] long l
);
/* opcode 0x02 */
long function1 (
```

```
[in] long l
);
/* opcode 0x03 */
long function2 (
[in] char c,
[out] long * l
);
```

Le code généré par des décompilateurs ne s'appuyant par sur l'analyse du code binaire est le suivant :

```
/* opcode 0x00 */
long function_ph1 (
[in] char c,
[out] long * l
);
/* opcode 0x01 */
long function_ph2 (
[in] long l
);
```

Les fonctions réellement intéressantes ne ressortent pas dans le code généré. Cette technique d'obscurcissement devient vraiment efficace avec plus de 2 fonctions RPC. Cette particularité est du au fait que la chaîne de format réutilise le même code pour des fonctions identiques. Or il est impossible de détecter cela sans une analyse du code binaire.

4 Une sécurité pour le moins douteuse

4.1 Vulnérabilités relevant de la conception

Certains choix ont été effectués par Microsoft dans la conception de leurs RPC. Si certains n'ont eu que des conséquences ennuyeuses, d'autres se sont révélés réellement catastrophiques.

Partage des interfaces et points finaux RPC dans un même processus Pour des raisons qui nous sont étrangères, Microsoft a doté son mécanisme RPC d'une propriété particulière, à l'origine de bien des maux. Tous les points finaux et interfaces sont « partagés » au sein d'un même processus : cela signifie qu'il est possible d'accéder à tout sous-ensemble de fonctions distantes à partir de n'importe quel point final disponible dans un processus.

Cela permet de contourner les restrictions d'accès posées sur un point final, de s'affranchir d'authentification, d'accéder à des fonctions initialement exposées seulement localement.

Principe du moindre privilège et divulgation d'information Le principe du moindre privilège dont il est ici question n'est évidemment pas la doctrine sécuritaire maintenant largement connue mais une adaptation libre de la part de Microsoft. Dès NT 4.0, l'éditeur a opté pour une « transparence » maximale de son système d'exploitation vis-à-vis d'éventuels clients réseaux. En effet, de nombreuses fonctions distantes sont accessibles par défaut dans Windows et ne nécessitent qu'un moindre privilège : un identifiant et un mot de passe nuls (aussi appelé « Null Session »).

Les informations que l'on peut en tirer devraient faire blêmir tout administrateur un tant soit peu consciencieux : version du système d'exploitation, régionalisation, utilisateurs, groupes, privilèges, processus, ...

Le principe de partage des interfaces et points finaux facilite d'autant le travail à un éventuel utilisateur malicieux. Voici quelques exemples connus ou non : Afin de mieux comprendre comment cela fonctionne nous allons étudier

Bulletin	Endpoint original	Endpoint d'accès	Fonction RPC	Description Description
URP1	\pipe\svcctl	\pipe\srvsvc	EnumServiceStatus	Liste les services
URP1	\pipe\eventlog	\pipe\srvsvc	OpenEventLog	Accès aux logs système
0day (XP SP1)	\pipe\Ctx_Winstation_API_Service	\pipe\srvsvc \pipe\srvsvc	RpcWinstation OpenServer	Liste les process

la « faille » concernant « Terminal Service » sur Windows XP SP1. Lorsque ce service est activé l'interface RPC suivante est disponible :

```
pipe: Ctx_Winstation_API_Service, uuid:
5ca4a760-ebb1-11cf-8611-00a0245420ed v1.0
```

Il est cependant impossible d'accéder à ce service sans disposer d'un mot de passe valide. Cependant une autre interface est disponible en anonyme :

```
pipe: srvsvc, uuid: 5ca4a760-ebb1-11cf-8611-00a0245420ed v1.0
```

Or il est possible d'accéder à l'interface du service « Terminal Service » en passant par le service « srvsvc » et ainsi échapper aux restrictions d'accès :

```
fid = bind_pipe (pipe:"$\backslash $srvsvc",
uuid:"5ca4a760-ebb1-11cf-8611-00a0245420ed", vers:1);
```

En utilisant cette méthode il est possible d'appeler la fonction RpcWinstationOpenServer et ainsi d'énumérer la liste des process actifs.

4.2 Vulnérabilités relevant de l'implémentation

Au cours de ces dernières années, les RPC sont devenus le premier vecteur d'exécution de code à distance, sans interaction de l'utilisateur, sur les systèmes d'exploitation Microsoft. La raison en est simple : la quantité de fonctionnalités accessibles anonymement, ou après authentification minimale, à distance grâce aux RPC est importante, et par conséquent le code que cela représente aussi. Ce dernier, assurément, n'est pas exempt de bogues.

Il nous semble opportun de rappeler ici les fonctions distantes ayant par le passé présenté des vulnérabilités :

Bulletin	Module	Identifiant de l'interface	Fonction
MS05-051	MsDtc	906b0ce0-c70b-1067-b317-00dd010662da	BuildContextW
MS05-043	Spoolss	12345678-1234-abcd-ef00-0123456789ab	AddPrinterExW
MS05-046	NwWks	e67ab081-9844-3521-9d32-834f038001c0	NwrGetUser NwrGetUser
MS05-039	UmPnpMgr	8d9f4e40-a03d-11ce-8f69-08003e30051b	PNP_Detect ResourceConflict
MS05-017	Msmq	fdb3a030-065f-11d1-bb9b-00a024ea5525	QMDeleteObject
MS05-010	Lls	342cfd40-3c6c-11ce-	LlsReplication
		a893-08002b2e9c6d	RequestW
MS04-011	Lsarpc	3919286a-b10c-11d0-9ba8-00c04fd92ef5	LsarClear AuditLog
MS03-039	epmapper	000001a0-0000-0000-c000-000000000046	RemoteGet ClassObject
MS03-026	epmapper	4d9f4ab8-7d1c-11cf-861e-0020af6e7c57	Remote Activation

4.3 Quelques « fonctionnalités » surprenantes (Exemples de rétro-conception d'interfaces RPC)

Le service client DNS Avec Windows 2000, Microsoft a introduit un service de résolution DNS avec cache, exposant une interface RPC localement (donc une interface LPC). Un rapide coup d'œil à la bibliothèque dynamique impliquée permet d'isoler l'interface, et les symboles nous renseignent sur les fonctions accessibles au travers de cette dernière. Leurs noms sont explicites, par exemple :

- CRrReadCache,
- CRrReadCacheEntry,
- CrrCacheRecordSet.

Cela devient intéressant lorsque l'on s'aperçoit que ce service est hébergé au sein du binaire services.exe qui abrite de nombreux autres interfaces et points finaux RPC, dont certains accessibles anonymement à distance. Grâce aux propriétés de partages abordées précédemment, il devenait possible de contrôler le cache DNS d'un poste Windows 2000 à distance sans aucune authentification.

Le préfixe IDL size_is Comme nous l'avons vu précédemment le langage IDL permet d'utiliser le préfixe « size_is » pour définir la taille réelle d'un tableau transmis en argument :

```
[in][size_is (120)] char * tab;
```

Le code précédent représente un tableau de caractères de taille 120. On peut cependant trouver ce préfixe dans des conditions quelques peu différentes :

```
/* opcode: 0x30, address: 0x7509E017 */
long _NetrDfsCreateExitPoint (
[in][unique][string] wchar_t * arg_1,
[in] struct struct_17 * arg_2,
[in][string] wchar_t * arg_3,
[in] long arg_4,
[in] long arg_5,
[out][size_is(arg_5)] wchar_t * arg_6
);
```

La fonction NetrDfsCreateExitPoint est présente dans l'interface RPC « srvsvc ». Grâce (ou à cause) du partage des interfaces, « srvsvc » est accessible en anonyme via l'interface « browser ». On peut donc appeler la fonction précédente sans avoir besoin d'un mot de passe valide. Or la chaîne unicode arg_6 est un argument qui n'est présent que dans la réponse. De plus, cet argument utilise le préfixe size_is avec l'argument arg_5 qui est lui dans la requête.

L'implémentation RPC de Windows restreint cependant le champ size_is à 2 Gigas (0x7FFFFFFF). De plus on est ici dans le cas d'une chaîne unicode. Il faut donc que l'argument arg_4 soit inférieur à 1 Giga.

Un attaquant peut donc appeler cette fonction en anonyme et forcer la machine distante à allouer jusquà 1 Giga octet de mémoire, se qui aura pour effet de saturer la machine cible durant quelques secondes. Il suffit alors de répéter la requête (quelques octets seulement) pour provoquer un déni de service.

Une autre caractéristique très surprenante du préfixe size_is concerne les stubs en ligne. Soit le code suivant :

```
void test (
[in] long arg_1,
[in][size_is(arg_1)] char * tab
);
```

Si la fonction RPC est compilée en stub interprété alors le champ arg_1 sera testé pour vérifier que sa valeur est comprise entre 0 et 0x7FFFFFFF. Or ce

n'est pas le cas avec un stub en ligne ! Il est donc tout à fait possible d'envoyer un tableau de taille 100 avec l'argument arg_1 mis à 0xFFFFFFFF. Or si le code de la fonction test se base sur la valeur de arg_1 pour allouer une zone de mémoire équivalente à celle du tableau, il devient envisageable de pouvoir produire un déni de service, voir un heap overflow.

Il est donc par conséquent très important de ne pas se fier totalement à la définition IDL mais de continuer à faire les tests nécessaire dans le code pour se prévenir de se genre de failles.

5 Des efforts qui vont dans le bon sens

Bien que les différents points soulevés précédemment soient alarmants, Microsoft a réagi et apporte au fur et à mesure des corrections aux problèmes liés aux RPC, pas seulement en publiant des correctifs, mais en modifiant leur système d'exploitation.

En effet, l'arrivée de Windows XP SP2 et Windows 2003 SP1 a considérablement réduit le nombre de services accessibles en anonyme ainsi que le nombre d'interfaces partagées au sein d'un même process. Par exemple, sous Windows XP SP2, seul les interfaces « browser » et « spoolss » restent accessibles en anonyme ainsi que l'interface « srvsvc » en utilisant « browser ». La restriction du nombre d'interfaces accessibles sans mot de passe réduit donc considérablement l'exploitation massive d'une faille RPC potentielle.

Un autre changement important lié à Windows XP SP2 et Windows 2003 SP1 réside dans le code même des fonctions RPC. En effet toutes les fonctions considérées comme dangereuses (printf, strcpy, strcat) ont été remplacées par des fonctions plus sûres (snprintf, strncpy, ...).

Un audit de sécurité approfondi à également été effectué afin de diminuer le nombre de failles pouvant être présentes. Il suffit, pour s'en convaincre, de regarder le nombre de failles récentes qui ont concernées Windows XP SP1 mais qui n'étaient pas présentes dans le SP2.

Enfin la sortie de URP1 pour Windows 2000 a également permis de diminuer le nombre de services RPC accessibles en anonyme mais a également grandement diminuer les informations qu'il était possible d'obtenir.

6 Microsoft, seul coupable ?

Bien que la majorité des failles RPC connues concerne le système d'exploitation Windows, d'autres failles toutes aussi critiques affectent d'autres logiciels. On peut, par exemple, présenter une faille qui affectaient les versions Windows du logiciel Veritas Backup Exec [VERITAS] :

```
93841fd0-16ce-11ce-850d-02608c44967b v1.0 / TCP port 6106
```

Ce service RPC offrait tout simplement, à quiconque se connectait sur le port TCP 6106, un accès total (lecture et écriture) à la base de registre Windows

et permettait donc à un attaquant de compromettre le système. Le correctif de sécurité supprime le service RPC. Cependant tout n'est pas encore complètement fixé ! (* 0day *).

7 Conclusion

Si Microsoft fait de gros efforts de sécurisation de ses fonctionnalités fondées sur RPC, il ne faut pas perdre de vue que les RPC sont utilisés par bon nombre de développeurs comme base de communication entre applications cliente et serveur. A ce titre, même si le cœur du système tend à s'assainir, les vulnérabilités présentes un jour dans le système d'exploitation ont toutes les chances de se retrouver à un moment ou un autre dans les applicatifs.

Références

[MIDL] MSDN, Microsoft Interface Definition Language, `http://msdn.microsoft.com/library/default.asp?url=/library/en-`us/dnanchor/html/midl.asp

[WNSI] Windows Network Services Internals, `http://www.hsc.fr/ressources/articles/win_net_srv/`

[MUDDLE] A MIDL format string disassembler, `http://www.cse.unsw.edu.au/~matthewc/muddle/`

[UNMIDL] MIDL decompiler in python, `http://www.immunitysec.com/resources-freesoftware.shtml`

[MIDA] MIDL decompiler plugin for IDA, `http://cgi.tenablesecurity.com/tenable/mida.php`

[VERITAS] VERITAS Backup Exec Server Remote Registry Access Vulnerability, `http://seer.support.veritas.com/docs/276605.htm`