Crackme 1 :

Cette épreuve se présente sous la forme d'un exécutable PE32 dans lequel les participants devront récupérer le flag en utilisant un debugger tel que OllyDbg. Si le participant n'arrive pas à déclencher l'affichage du flag par le programme, celui-ci affiche « Bad boyz ».

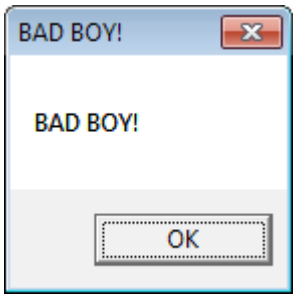

Illustration 1: Bad boy

A l'ouverture du programme avec OllyDbg on arrive sur du code généré automatiquement à la compilation, il ne représente donc pas un grand intérêt. En avançant avec F8 (Step over, permet de passer les appels de fonctions sans rentrer dedans.), on arrive rapidement sur l'appel du main du programme :

```
0040149B  > 51            PUSH ECX                          Arg4
0040149C  . 50            PUSH EAX                          Arg3
0040149D  . 56            PUSH ESI                          Arg2
0040149E  . 68 00004000   PUSH ctf3.00400000                Arg1 = 00400000
004014A3  . E8 18FDFFFF   CALL ctf3.004011C0                ctf3.004011C0
```

Illustration 2: Appel du main

On découvre le code assembleur de la fonction principale de notre programme :

```
004011C0  $ 55            PUSH EBP
004011C1  . 8BEC          MOV EBP,ESP
004011C3  . 83EC 10       SUB ESP,10
004011C6  . 8D45 F0       LEA EAX,DWORD PTR SS:[EBP-10]
004011C9  . 50            PUSH EAX                          pSystemTime
004011CA  . FF15 04804000 CALL DWORD PTR DS:[<&KERNEL32.GetSystem GetSystemTime
004011D0  . 0FB74D F0     MOVZX ECX,WORD PTR SS:[EBP-10]
004011D4  . 890D 48C84000 MOV DWORD PTR DS:[40C848],ECX
004011DA  . 8B15 48C84000 MOV EDX,DWORD PTR DS:[40C848]
004011E0  . C1E2 08       SHL EDX,8
004011E3  . 8915 48C84000 MOV DWORD PTR DS:[40C848],EDX
004011E9  . 0FB645 F2     MOVZX EAX,BYTE PTR SS:[EBP-E]
004011ED  . 0305 48C84000 ADD EAX,DWORD PTR DS:[40C848]
004011F3  . A3 48C84000   MOV DWORD PTR DS:[40C848],EAX
004011F8  . 8B0D 48C84000 MOV ECX,DWORD PTR DS:[40C848]
004011FE  . C1E1 08       SHL ECX,8
00401201  . 890D 48C84000 MOV DWORD PTR DS:[40C848],ECX
00401207  . 0FB655 F6     MOVZX EDX,BYTE PTR SS:[EBP-A]
0040120B  . 0315 48C84000 ADD EDX,DWORD PTR DS:[40C848]
00401211  . 8915 48C84000 MOV DWORD PTR DS:[40C848],EDX
00401217  . A1 48C84000   MOV EAX,DWORD PTR DS:[40C848]
0040121C  . 50            PUSH EAX                          Arg1 => 07DE0910
0040121D  . E8 BEFEFFFF   CALL ctf3.004010E0                ctf3.004010E0
00401222  . 83C4 04       ADD ESP,4
00401225  . 8D05 20BB4000 LEA EAX,DWORD PTR DS:[40BB20]
0040122B  . FFD0          CALL EAX
0040122D  . 813D 20BB4000 CMP DWORD PTR DS:[40BB20],83EC8B55
00401237  .v74 01         JE SHORT ctf3.0040123A
00401239  . CC            INT3
0040123A  > 33C0          XOR EAX,EAX
0040123C  . 8BE5          MOV ESP,EBP
0040123E  . 5D            POP EBP
0040123F  . C2 1000       RETN 10
```

Illustration 3: Fonction principale

On remarque que le temps du système est récupéré, suivit par quelques calculs sur celui-ci, que le résultat est transmit à une fonction et enfin l'appel de la fonction 0x40BB20. Cette adresse est intrigante car elle ne se situe pas dans la section .code mais dans la section .data

```
00400000 00001000 ctf3          PE header  Imag R  RWE
00401000 00007000 ctf3  .text   code       Imag R  RWE
00408000 00003000 ctf3  .rdata  imports    Imag R  RWE
0040B000 00002000 ctf3  .data   data       Imag R  RWE
0040D000 00001000 ctf3  .rsrc   resources  Imag R  RWE
```

Illustration 4: Liste des sections

En exécutant le programme une première fois, on obtient « Bad boyz » mais pourtant on ne voit pas d'appel à une MessageBox dans la fonction main ... On utilisant la fonction que l'on retrouve en effectuant un clique droit -> Search for -> All referenced text strings, on retrouve ou se situe l'appel de la MessageBox

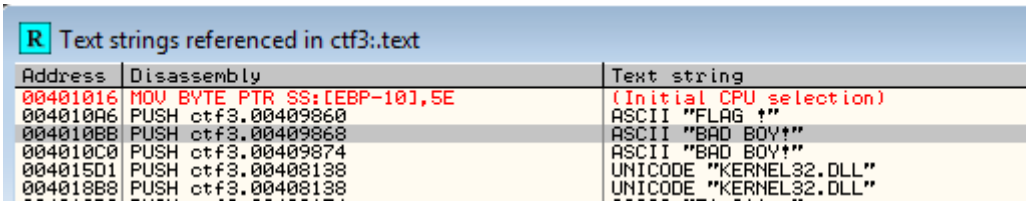

Illustration 5: All regerenced text strings

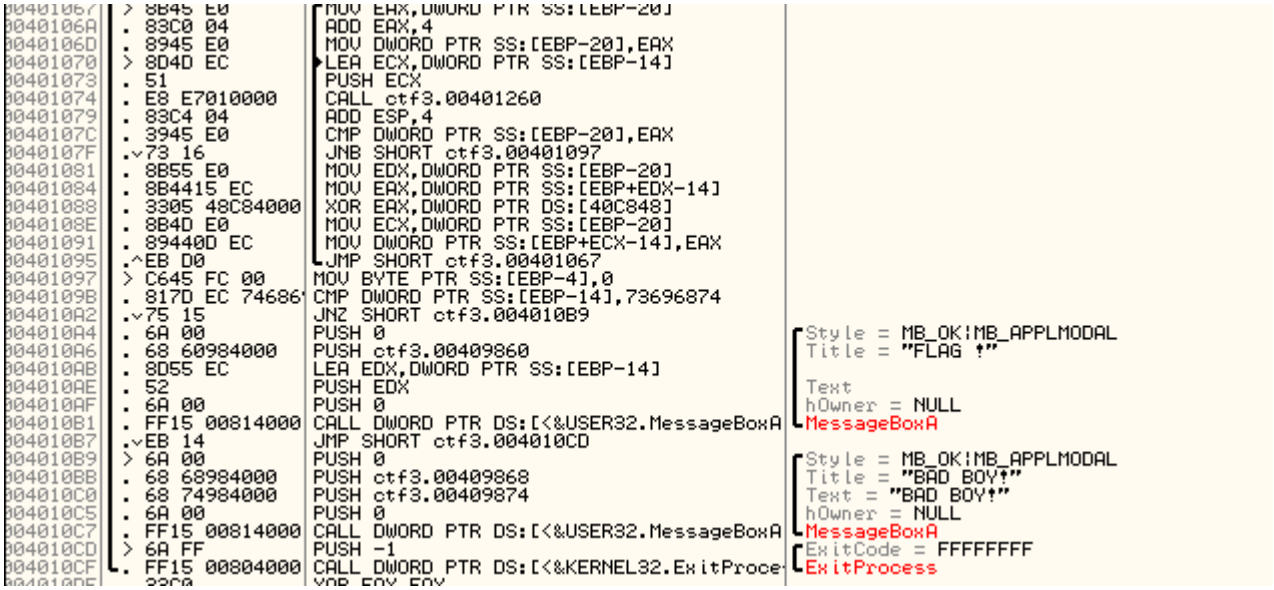

Illustration 6: Fonction qui appelle la MessageBox

Dans un premier temps, il effectue des opérations sur une zone mémoire avec le temps précédemment calculé, puis il compare les 4 premiers octects de cette zone avec « 0x73696874 » ce qui correspond au texte « this ». Mais cette fonction est un leurre et en calculant la bonne clef on obtient le message « thisifnottheflag ». L'opération étant un simple xor, on calcul la clef avec le procédé suivant : Les quatres premiers octets du buffer valent le int 0x736A7B43, si on xor avec « this » qui vaut 0x73696874 on obtient la clef 0x31337. En patchant ou en posant un breakpoint pour modifier la valeur de la variable globale contenant le temps on obtient le message suivant :

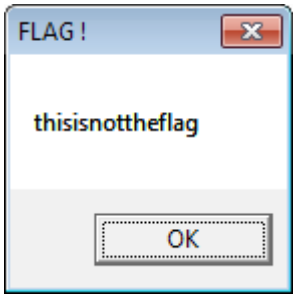

Illustration 7: Faux flag

Illustration 8: Patch du faux flag

Donc cette fonction était un leurre, mais on ne sait toujours pas comment elle est appelé, en cherchant des références à son adresse on tombe sur l'initialisation d'un gestionnaire d'exception qui va prendre le contrôle si le programme plante (violation d'espace mémoire protégé, division par zéro, etc ..)

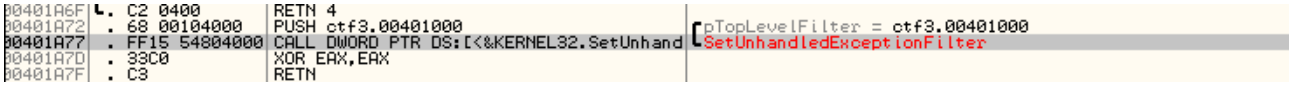

Illustration 9: Initialisation du SEH

En revenant sur notre fonction principale, on remarque que les instructions de la fonction 0x40BB20 n'ont pas de sens car celles-ci sont déchiffrés avec une mauvaise clef (calculée à partir du temps). Elles vont faire planter le programme et passer la main à notre fonction contenant la faux flag. Il y'a également un saut conditionnel qui vérifie que les 4 premiers octets de la fonction 0x40BB20 sont 0x83EC8B55 mais cette fois si ce n'est pas du texte mais les instructions (il manque EC et 10 puisque l'ont compare que 4 octets):
55 PUSH EBP
8BEC MOV EBP,ESP
83EC 10 SUB ESP,10
Si ce n'est pas le cas le programme exécute l'instruction « int 3 » qui représente un breakpoint, mais utilisé hors d'un debuggeur cela fait planter le programme, appelant ainsi la fonction du faux flag. Cette condition permet d'être sur que la fonction du faux flag est appelée tant que le participant n'a pas la bonne clef.
Il reste au participant à bruteforcer la clef et de patcher le programme (ou modifier l'heure de son système ...) pour récupérer le flag. Le participant doit reverser l'algorithme pour être capable de le bruteforcer, il peut se servir de CFF explorer pour avoir un tableau contenant le buffer de la fonction à déchiffrer.

```
#include <Windows.h>
#include <stdio.h>


int main(){
	unsigned char data[185] = { // Buffer contenant la fonction à déchiffrer
	0x5F, 0xFF, 0xE8, 0x56, 0xFC, 0x8F, 0xDA, 0xD8, 0x02, 0xA6, 0xF9, 0x0D, 0x5E, 0xF2, 0xD1, 0x67,
	0x3A, 0xD3, 0x9E, 0x1C, 0x99, 0x4A, 0x99, 0x09, 0x63, 0x2F, 0x23, 0xB1, 0x33, 0xF1, 0xBA, 0xE6,
	0x2A, 0x15, 0x3C, 0x1D, 0x7F, 0x61, 0x92, 0x99, 0xC8, 0xCA, 0x65, 0x1F, 0xC0, 0x89, 0xBD, 0x30,
	0x19, 0x45, 0x17, 0x72, 0x1E, 0x0A, 0xDE, 0x77, 0xBE, 0x87, 0xB2, 0x4C, 0xEF, 0x30, 0x2E, 0x44,
	0x9D, 0x05, 0x43, 0x06, 0x4C, 0xB0, 0x5D, 0x72, 0x33, 0x8B, 0x7D, 0xEA, 0x5C, 0x19, 0x3A, 0xED,
	0x8F, 0x02, 0xAB, 0xBC, 0x7F, 0x6D, 0xAD, 0x70, 0x85, 0x27, 0xA2, 0xAC, 0x71, 0x3F, 0xB9, 0x30,
	0xFE, 0x5E, 0x86, 0x67, 0x0B, 0x95, 0xEA, 0x3D, 0xFE, 0x43, 0x81, 0xF9, 0xB4, 0xCF, 0x9E, 0x44,
	0x36, 0xE3, 0x81, 0x20, 0x42, 0x9A, 0x2B, 0x85, 0xC3, 0xD4, 0x0E, 0xA7, 0x5C, 0xDF, 0xDA, 0x8D,
	0x7F, 0xAE, 0x47, 0x2F, 0x16, 0xA6, 0x7B, 0x9B, 0xB3, 0x40, 0xD3, 0x9F, 0xA1, 0xF0, 0x8F, 0x01,
```

```
	0x6D, 0xC3, 0xC4, 0x31, 0x87, 0x5D, 0x18, 0xE2, 0x72, 0xFE, 0x07, 0xD8, 0x20, 0x6B,
0xB0, 0x34,
	0xB7, 0xC4, 0x97, 0xEC, 0xCB, 0x3E, 0xF5, 0x25, 0xE7, 0xE7, 0x99, 0x47, 0x99, 0x04,
0x8D, 0x6B,
	0xA2, 0x95, 0xDF, 0x63, 0x41, 0x41, 0xEE, 0xBA, 0x68
    };
	unsigned char data2[185];
	unsigned char dataRandom[185];
	unsigned int date=0;
	BYTE jour=0;
	BYTE mois = 0 ;
	WORD annee = 0;
	for(unsigned int i=0;i<0xFFFFFFFF;i++)
	{
		memcpy(data2,data,185);
		srand(i);
		for(int j=0;j<185;j++)
			dataRandom[j] = rand() % 256;
		for(int j=0;j<185;j++)
			data2[j]=dataRandom[j]^data2[j];
		if(*(int*)data2==0x83EC8B55)
		{//Push ebp mov ebp,esp + une partie de sub esp,xx
			printf("%x ",i);
			date=i;
			jour=(BYTE)date & 0XFF;
			date>>=8;
			mois=(BYTE)date & 0xFF;
			date>>=8;
			annee=(WORD)date & 0xFFFF;
			printf("%d/%d/%d\n",jour,mois,annee);
		}

	}

	system("pause");
	return 0;
}
```

Pour ne pas que le bruteforce ne dur trop longtemps, j'ai utilisé une propriété intéressante de la bibliothèque C du compilateur Visual. En effet la fonction rand de visual ne possède une difficulté d'uniquement 2^24 si on fait un module%256 sur sa sortie.
Donc, rand() % 256 va générer le même résultat pour tous les seed comprit entre [00XXXXXX] et [FFXXXXXX], exemple dans notre cas : [00DC0C16] à [FFDC0C16] génère les mêmes chiffres et permettent de valider le challenge.

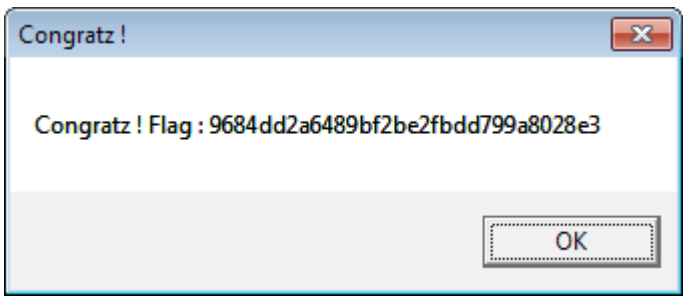

Illustration 10: Flag

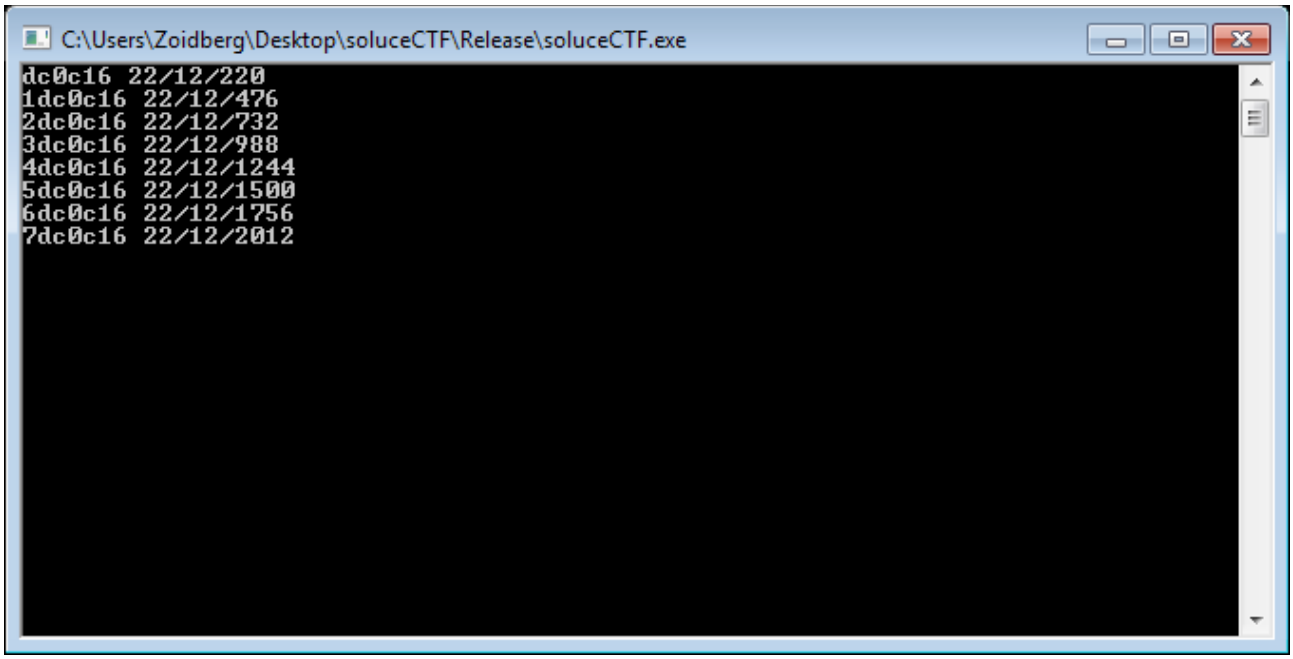

Illustration 11: Les différentes solutions

Le participant peut patcher le programme ou mettre son système à une date valide comme 22/12/2012