COMPRENDRE
forwarder.exe
securitech 2005

défi proposé par Nicolas Brulez
Solution rédigée par BeatriX^FRET

SOMMAIRE

1 . INTRODUCTION 3

2. SCHEMA DU CRACKME 4

3. ANTI-DESASSEMBLAGE 5

4. SEHS : ANTI HARDWARE BP / ANTI-TRACING 6

4.1 Comment contourner l'anti-tracing des SEHs ? 8

5. RIPEMD-128 10

6. APLIB SUR SECTION CODE 11

7. DECRYPTAGE DE LA SECTION DE DONNEES : FAILLE ! 12

8. REMERCIEMENTS 13

1. INTRODUCTION

Je vous propose dans cet article d'étudier les points principaux qui rendent ce défi intéressant. Je ne parlerai pas ici de degré de difficulté mais plutôt de l'état d'esprit de ce crackme. L'idée est assez bonne me semble-t-il et l'auteur a respecté le cadre fixé par le securitech 2005. Pour résumer en quelques mots ce défi, je dirais que l'auteur a réalisé une situation qui peut s'avérer réaliste. Il redémontre et exploite ici l'une des régles d'or de la protection de binaires : LA REGLE DU MAILLON LE PLUS FAIBLE. Niels Ferguson, dans son ouvrage « CRYPTOGRAPHIE EN PRATIQUE » le rappelle également : « **Un système de sécurité ne sera jamais plus résistant que son maillon le plus faible** ». Nicolas Brulez a bien évidemment laissé un maillon très faible dans son binaire et suffisamment évident pour qu'il soit exploitable rapidement dans le cadre du challenge. J'ai personnellement apprécié ce petit défi qui m'a permis de me remettre ce concept fondamental en mémoire (oui, je me suis fait avoir au premier abord).

Tout d'abord, voici la note qui était fournie avec ce binaire lors du challenge securitech2005.

Challenge 7: Reverse sur le relais

Lieu	Salle de briefing du DRI
Date	Samedi 18 Juin 2005 / 16h12
Intervenants	Patrick De Touvière : Directeur du DRI Arnaud Berge : RSSI du DRI Pierre Legard : Responçable du SR BeatriX: Agent du SR

Pierre Legard

Nos services ont fini de scanner le disque dur de la machine. Tout semble confirmer qu'il s'agit bien de l'ordinateur d'un particulier que les trafiquants ont piraté et qu'ils utilisent maintenant comme relais.

Patrick De Touvière

Bon, laissons ces détails techniques. Pouvez-vous remonter aux pirates ?

BeatriX

Pas immédiatement. L'accès à la backdoor ne nous donne pas l'adresse IP de ceux qui l'utilisent. De plus, elle s'exécute en tant qu'utilisateur non privilégié. Nous ne pouvons donc pas mettre en place directement un sniffer pour capturer l'adresse IP des pirates.

Arnaud Berge

Il vous suffit de passer administrateur.

BeatriX

C'est une opération qui n'est pas évidente sur un système bien patché et trop délicate : elle pourrait arrêter le système ou, pire, générer des traces qui alerteraient les pirates.

Ils n'auraient alors plus qu'à couper le lien avec ce relais.
Mais il existe une autre solution : « Stealth » n'est pas faite pour relayer des paquets. Les pirates ont donc utilisé un autre outil pour cela, et c'est cet outil qui écoutait sur l'autre port. Nous pensons qu'il est conçu pour accepter des requêtes uniquement d'une certaine adresse IP.

Arnaud Berge

Celle des pirates ! Leur adresse IP serait donc codée dans l'exécutable !

BeatriX

Oui, mais ils ont apparemment prévu la possibilité que nous récupérions ce programme : il semble bien protégé.

Patrick De Touvière

BeatriX, chargez-vous de contourner ces protections et de récupérer cette adresse.

Mission

Objectif

Récupérer l'adresse IP que le programme accepte de relayer

2. SCHEMA DU CRACKME

Ok Pat' ☺ on va te retrouver cette IP ! Afin d'être plus clair dans mon exposé, je vais illustrer le comportement du crackme. Vous pouvez suivre le déroulement des opérations sur la figure 1 :

1) Le crackme est protégé contre le débuggage et perturbe OllyDbg dès le début. Nous verrons qu'il s'agit d'une technique classique utilisée par Nicolas Brulez dans de précédents défis.

2) Le crackme est crypté par de nombres layers. Ceci rend tout désassemblage a priori impossible.

3) Chaque layer est accompagné d'une exception gérée par un SEH anti-Harware Break Points et anti-tracing.

4) Le crackme recherche un paramètre transmis en ligne de commande à l'aide de la fonction GetCommandLineA.

5) Le crackme utilise ce paramètre pour calculer un hash à l'aide de l'agorithme RIPEMD-128.

6) Le crackme utilise une partie de ce hash pour décrypter la section des données du binaire d'origine.

7) Le crackme décrypte la section du code du bainaire d'origine à l'aide de l'ApLib.

8) La suite du crackme ne nous intéresse pas ici puisque nous disposons de tous les renseignements pour trouver l'IP.

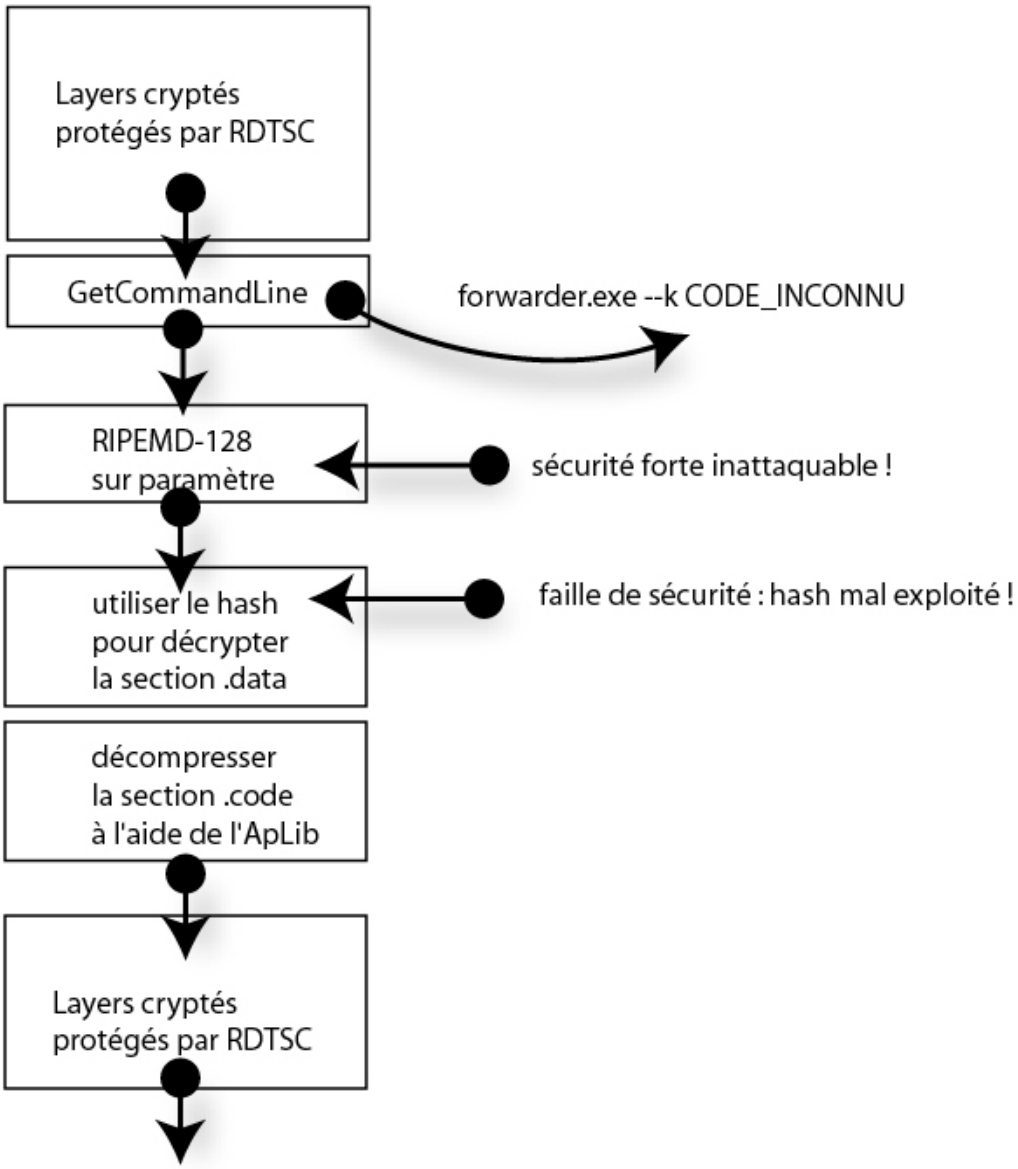
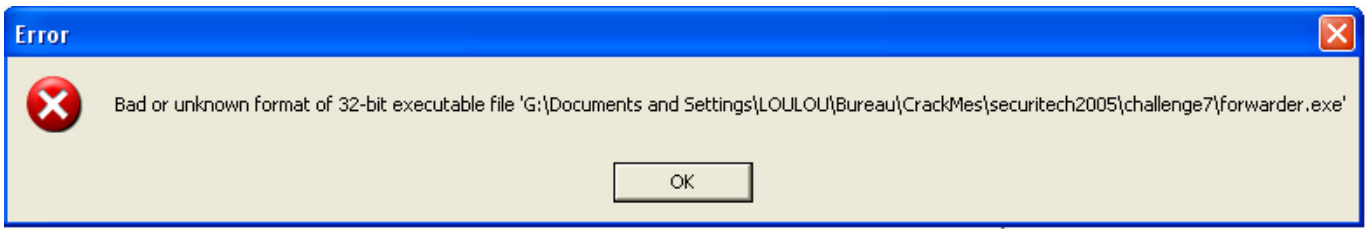


FIGURE 1

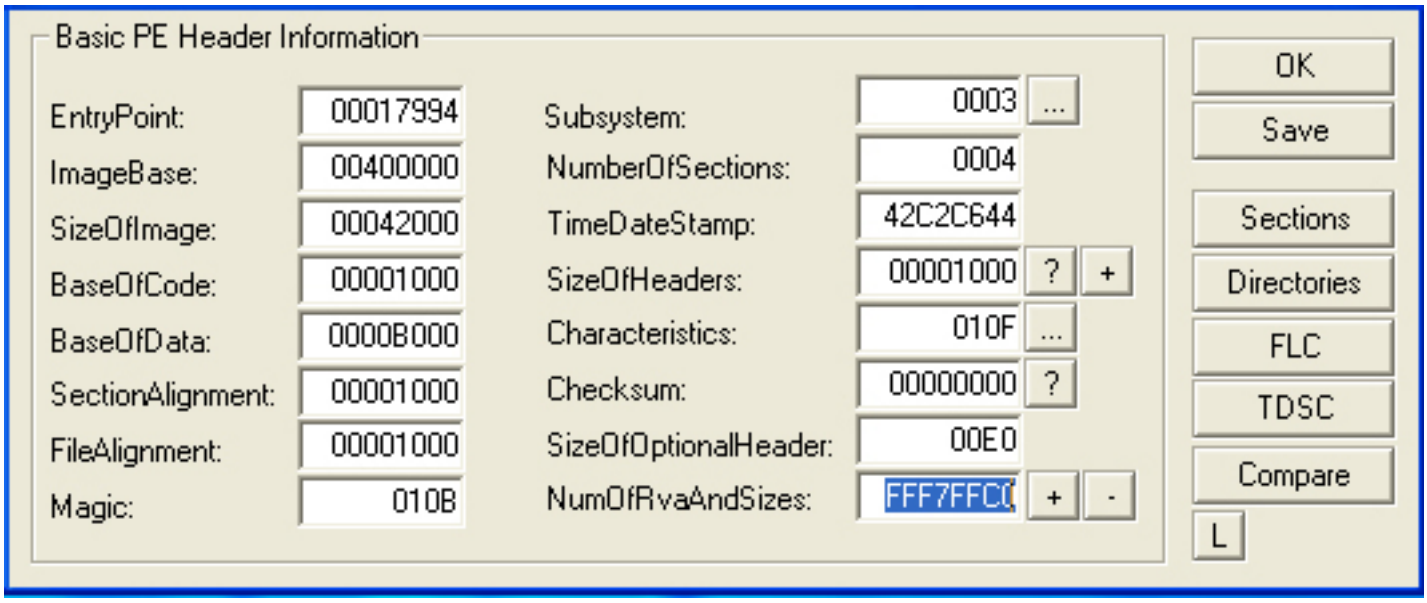
3. ANTI-DESASSEMBLAGE

Pour commencer, si vous tentez de désassembler le binaire forwarder.exe avec OllyDbg, voilà ce qui vous attend :



Pour les curieux, LOULOU, c'est moi ☺.

OllyDbg n'est pas planté à ce moment là mais il ne parvient pas à poser un BP sur l'ENTRY POINT ce qui peut dérouter facilement un novice. L'astuce de ce « trick » est à chercher du côté du header de ce binaire. Si on ouvre forwarder.exe à l'aide de LordPE, on voit ceci :



On remarque que le champ NumOfRvaAndSizes, autrement dit, le nombre de DIRECTORIES a une valeur bien étrange : FFF7FFC0. il suffit de passer cette valeur à 10h et le tour est joué. OllyDbg est alors capable de désassembler et surtout de breaker à l'Entry Point.

4. SEHS : ANTI HARDWARE BP / ANTI-TRACING

Utiliser les SEH (Structured Exception Handling) pour dérouter le débuggage est une technique très ancienne mais toujours efficace. Elle permet d'atteindre deux objectifs : Le principal, écraser les Debugs Registers en accédant au CONTEXT du processus. Le secondaire, dérouter les reversers qui ne maîtrisent pas les SEHS.

J'ai reconstitué la mise en place de tels SEHs et des exceptions qui vont avec. Il s'agit en fait d'un code relogeable utilisable à volonté. Le crackme génère deux types d'exceptions : la première est une exception générée en exécutant l'instruction UD2 et la seconde en exécutant INT1. Voici ce qu'on obtient sur la doc INTEL au sujet de UD2 :

UD2—Undefined Instruction

Description

Generates an invalid opcode. This instruction is provided for software testing to explicitly

generate an invalid opcode. The opcode for this instruction is reserved for this purpose.
Other than raising the invalid opcode exception, this instruction is the same as the NOP instruction.

Operation

#UD (* Generates invalid opcode exception *);

Flags Affected

None.

Exceptions (All Operating Modes)

#UD Instruction is guaranteed to raise an invalid opcode exception in all operating modes.

Opcode Instruction Description

0F 0B UD2 Raise invalid opcode exception.

Il s'agit donc d'une instruction qui génère (exprès !) une exception. Elle est donc gérée par un SEH. Le handler de ce SEH va s'occuper des DRx pour écraser les HBP mais va aussi tester la durée écoulée (en cycles). Si le binaire est sous débugger, le « temps » sera plus long qu'une valeur seuil et le crackme va alors réagir en conséquence. Voici le code (c'est sensiblement le même dans le cas de INT1) :

```
.586P
.model flat, stdcall
option casemap:none
assume fs: nothing
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib


.code
ud2 MACRO
  BYTE    0Fh, 0Bh
ENDM

start:
  pushad
  call $+4Dh                        ; pousse l'adresse du handler sur la pile

; ****************************** Handler
  mov ecx, dword ptr [esp+0Ch]
  add dword ptr [ecx+0B8h],2        ; Modifie EIP
  xor eax, eax
  mov dword ptr [ecx+4], eax        ; Ecrasement des DRx
  mov dword ptr [ecx+8], eax
  mov dword ptr [ecx+0Ch], eax
  mov dword ptr [ecx+10h], eax
  mov dword ptr [ecx+14h], eax
  mov dword ptr [ecx+18h], 155h
  mov eax, dword ptr [ecx+0B0h]     ; Récupère EAX au moment de l'exception
  push eax
  cpuid
  rdtsc
  sub eax, dword ptr [esp]          ; Calcul l'écart de temps écoulé entre les deux RDTSC
  add esp,4
  cmp eax, 0E0000h                  ; Compare cet écart à une valeur seuil
  ja @F
  xor eax, eax
  ret
@@:
  add dword ptr [ecx+0B8h], 36h     ; Modifie encore EIP en cas de temps trop grand
  xor eax, eax
  ret
; ********************** génère une exception
```

```
xor eax, eax
push dword ptr fs:[eax]
mov dword ptr fs:[eax], esp
cpuid
rdtsc
ud2                                    ; Génère une exception
pop dword ptr fs:[0]
add esp,4
popad

push 0
call ExitProcess

end start
```

Comme vous pouvez le constater, si le temps entre les deux RDTSC (celui avant l'exception et celui du handler) est supérieur à la valeur seuil E0000h, alors, le handler ajoute 36h à EIP.

Remarque : Sur mon système (XP+SP1), le programme se plante dans le handler. Je m'explique : Sous OllyDbg, le temps entre les deux RDTSC est toujours supérieur à E0000h. Le handler réagit donc en conséquence et essaie d'ajouter 36h à EIP. Or, il essaie d'atteindre EIP à en faisant [ECX+0B8h] et en supposant que ECX soit l'adresse du CONTEXT. Malheureusement, l'instruction CPUID qui précède RDTSC modifie ECX et du coup, il ne pointe plus vers le CONTEXT et c'est le crash. Ceci n'a pas beaucoup d'importance en réalité mais vous pouvez vous retrouver confronté à ce problème.

4.1 Comment contourner l'anti-tracing des SEHs ?

Le problème reste entier : crash ou pas crash, les handlers nous mettent en vrac à tous les coups en modifiant l'EIP de façon erronée. Ce qu'il nous faut, c'est modifier les handlers eux-mêmes avant qu'ils soient exécutés ! Il faut en fait intervenir entre l'exception générée et l'exécution du handler. Ce que je propose ici, c'est de nopper le JA qui suit le CMP EAX, E0000h dans tous les handlers utilisés. En fait, c'est faisable assez facilement en insérant un bout de code avant que le handler soit exécuté. Faites l'expérience ! Lancez le binaire, il va générer la première exception « grâce » à UD2. Si vous faites SHIFt+F7 (comme vous le propose gentiment Olly), vous tombez sur ce genre de code :

ilot atteint après SHIFT + F7

```
77FA4DB3  MOV EBX,DWORD PTR SS:[ESP]
77FA4DB6  PUSH ECX
77FA4DB7  PUSH EBX
77FA4DB8  CALL ntdll.77F50B69
77FA4DBD  OR AL,AL
77FA4DBF  JE SHORT ntdll.77FA4DCD
77FA4DC1  POP EBX
77FA4DC2  POP ECX
77FA4DC3  PUSH 0
77FA4DC5  PUSH ECX
77FA4DC6  CALL ntdll.ZwContinue
77FA4DCB  JMP SHORT ntdll.77FA4DD8
77FA4DCD  POP EBX
77FA4DCE  POP ECX
77FA4DCF  PUSH 0
77FA4DD1  PUSH ECX
77FA4DD2  PUSH EBX
77FA4DD3  CALL ntdll.ZwRaiseException
```

```
77FA4DD8  ADD ESP,-14
77FA4DDB  MOV DWORD PTR SS:[ESP],EAX
77FA4DDE  MOV DWORD PTR SS:[ESP+4],1
77FA4DE6  MOV DWORD PTR SS:[ESP+8],EBX
77FA4DEA  MOV DWORD PTR SS:[ESP+10],0
77FA4DF2  PUSH ESP
77FA4DF3  CALL ntdll.RtlRaiseException
77FA4DF8  RETN 8
```

Ceci est une véritable aubène ! Rendez vous compte : Le système nous redonne la main durant quelques lignes avant de lancer le handler du SEH ! Il suffit de détourner ces lignes vers une routine à nous qui va s'occuper personnellement du handler. Imaginons que notre routine s'installe en 40CA10h (de façon arbitraire), voici ce à quoi doit alors ressembler l'ilot précédent :

ilot atteint après SHIFT + F7 (modifié)

```
77FA4DB3  JMP forwarde.0040CA10
77FA4DB8  CALL ntdll.77F50B69
77FA4DBD  OR AL,AL
77FA4DBF  JE SHORT ntdll.77FA4DCD
77FA4DC1  POP EBX
77FA4DC2  POP ECX
77FA4DC3  PUSH 0
77FA4DC5  PUSH ECX
77FA4DC6  CALL ntdll.ZwContinue
77FA4DCB  JMP SHORT ntdll.77FA4DD8
77FA4DCD  POP EBX
77FA4DCE  POP ECX
77FA4DCF  PUSH 0
77FA4DD1  PUSH ECX
77FA4DD2  PUSH EBX
77FA4DD3  CALL ntdll.ZwRaiseException
77FA4DD8  ADD ESP,-14
77FA4DDB  MOV DWORD PTR SS:[ESP],EAX
77FA4DDE  MOV DWORD PTR SS:[ESP+4],1
77FA4DE6  MOV DWORD PTR SS:[ESP+8],EBX
77FA4DEA  MOV DWORD PTR SS:[ESP+10],0
77FA4DF2  PUSH ESP
77FA4DF3  CALL ntdll.RtlRaiseException
77FA4DF8  RETN 8
```

J'ai donc remplacé les trois premières instructions par un simple jump qui détourne le déroulement normal. En 40CA10, je place ma routine :

Modifier le handler avant son exécution

```
0040CA10  PUSHAD
0040CA11  MOV ECX,DWORD PTR FS:[0]
0040CA18  MOV ECX,DWORD PTR DS:[ECX+4]          ; Atteindre le handler
0040CA1B  CMP DWORD PTR DS:[ECX],310FA20F       ; chercher CPUID/RDTSC
0040CA21  JE SHORT forwarde.0040CA26
0040CA23  INC ECX
0040CA24  JMP SHORT forwarde.0040CA1B
0040CA26  ADD ECX,4
0040CA29  CMP BYTE PTR DS:[ECX],3D              ; Chercher le CMP suivant
0040CA2C  JE SHORT forwarde.0040CA31
0040CA2E  INC ECX
0040CA2F  JMP SHORT forwarde.0040CA29
0040CA31  ADD ECX,5
0040CA34  MOV WORD PTR DS:[ECX],9090            ; Nopper le JA
```

```
0040CA39  POPAD
0040CA3A  MOV EBX,DWORD PTR SS:[ESP]          ; Exécuter les instructions écrasées
0040CA3D  PUSH ECX
0040CA3E  PUSH EBX
0040CA3F  JMP ntdll.77FA4DB8                  ; revenir dans l'ilot détourné
```

Voilà ☺ Les handlers deviennent inoffensifs et nous pouvons avancer tranquillement dans le code. On peut par la même occasion empêcher l'écrasement des DRx mais ici, ce n'est pas vraiment utile.

5 . RIPEMD-128

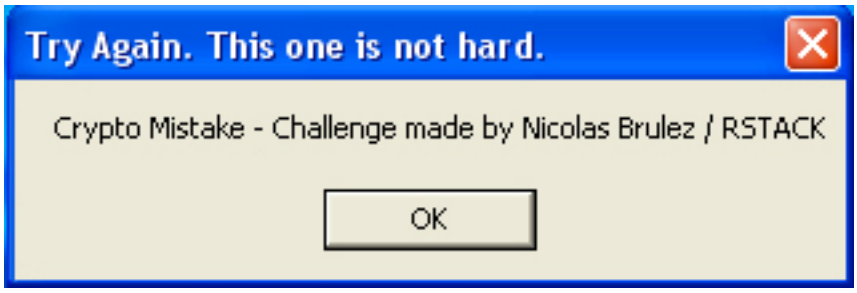
Après avoir franchi un certain nombre d'exceptions (nombreuses !), nous atteignons le code suivant :

RIPEMD-128 sur paramètre

```
00422250  PUSHAD
00422251  CALL DWORD PTR SS:[EBP+409A07]  <------------ GetCommandLineA
00422257  MOV EDI,EAX
00422259  MOV ECX,-1
0042225E  XOR AL,AL
00422260  PUSH EDI
00422261  REPNE SCAS BYTE PTR ES:[EDI]
00422263  NOT ECX
00422265  POP EDI
00422266  CMP DWORD PTR DS:[EDI],206B2D2D        < -------- Recherche la chaine « --k »
0042226C  JE SHORT forwarde.00422274

0042226E  INC EDI
0042226F  DEC ECX
00422270  JECXZ SHORT forwarde.004222A7
00422272  JMP SHORT forwarde.00422266


00422274  ADD EDI,4                            < --------- Se positionner sur le "SERIAL"
00422277  PUSH EDI
00422278  XOR ECX,ECX
0042227A  DEC ECX
0042227B  XOR AL,AL
0042227D  REPNE SCAS BYTE PTR ES:[EDI]
0042227F  NOT ECX
00422281  DEC ECX       <------------------------ taille du serial
00422282  JECXZ SHORT forwarde.004222A7

00422284  POP EBX
00422285  PUSH ECX <---------------------- taille du serial
00422286  PUSH EBX  <---------------------- pointe sur le serial
00422287  CALL forwarde.00420507  <----------------- RIPEMD-128
0042228C  MOV EBX,DWORD PTR DS:[EAX]
0042228E  MOV ESI,DWORD PTR DS:[EAX+C]
00422291  XOR EBX,ESI
00422293  MOV ESI,DWORD PTR DS:[EAX+8]
00422296  BSWAP ESI
00422298  CMP ESI,5E058AE4              < ----------- Teste le 3ème DWORD du hash
0042229E  JE forwarde.00422320
004222A4  ADD ESP,8
004222A7  POPAD
004222A8  PUSH 0
004222AA  CALL forwarde.004222D0  <----------------- EXIT
```

Le crackme commence par récupérer le paramètre (que j'appelle ici SERIAL) situé derrière « --k » placé en ligne de commande. Il calcule alors un hash (RIPEMD-128) accessible via EAX. On peut détecter le type d'algorithme en scannant un dump à l'aide du plugin de **PEID**. Ne faites pas comme moi : la recherche sur google des paramètres transmis pour l'algorithme indiquent a priori un MD5… évidemment, j'y ai cru un certain temps ☺

Si vous avez entré le bon paramètre, le test de la ligne 422298 se passe bien et le programme continue sinon, vous avez droit à cette jolie fenêtre :



Forcer un RIPEMD-128 est en fait un vrai suicide ! Pour exemple, si le SERIAL est constitué de 5 caractères, un BruteForce peut nous amener sur plus de 10 heures de recherche (si on n'a pas de chance). Vous imaginez bien que dans ce scénario, les pirates ont utilisé un « SERIAL » beaucoup plus long que l'on ne retrouvera jamais. J'ai cru un instant qu'on le pouvait, ce fût une grave erreur de ma part ! La faille n'est donc pas à chercher ici !

6 . APLIB SUR SECTION CODE

Si on va voir en 422320 (si le test du hash est positif), on obtient ceci :

décompression de la section code + décryptage de la section données

```
00422320  MOV EAX,forwarde.0040D000
00422325  MOV ECX,2FFC
0042232A  SUB DWORD PTR DS:[EAX+ECX],DEAD1337
00422331  ADD DWORD PTR DS:[EAX+ECX],ECX
00422334  XOR DWORD PTR DS:[EAX+ECX],EBX  <---------- EBX = 904DA2D6h
00422337  SUB ECX,4
0042233A  JNZ SHORT forwarde.0042232A
0042233C  ADD ESP,8
0042233F  POPAD
00422340  JMP forwarde.004223EE

004223EE  MOV EAX,forwarde.00401000
004223F3  PUSH EAX
004223F4  MOV EAX,forwarde.00412000
004223F9  PUSH EAX
004223FA  CALL forwarde.00422345 <---------- décrypte ApLib
```

Nous tombons sur deux routines. La première est chargée de décrypter la section 40D000h, section de données du binaire d'origine. Lé seconde est chargée de décompresser la section 401000h, section de code de la backdoor. Si on dump la section de code (401000h) après décompression à l'aide de l'ApLib, on constate (étude statique), que la backdoor est composée de deux parties : une partie client et une partie serveur. Or, nous savons que ce binaire a été

retrouvé sur l'ordinateur d'une victime, il est donc utilisé en tant que serveur pour se connecter sur l'IP des pirates. Il nous faut donc analyser la portion SERVEUR de ce binaire que l'on trouve très vite en 401500 :

SERVEUR de la BackDoor

```
push    ebp
mov     ebp, esp
sub     esp, 198h
mov     dword ptr [ebp-4], 0
mov     word ptr [ebp-198h], 2
lea     eax, [ebp-194h]
push    eax
mov     cx, [ebp-198h]
push    ecx
call    WSAStartup
push    offset dword_40D1A4          ; Chaine : adresse IP relayée par le programme
call    inet_addr
mov     [ebp-4], eax
mov     edx, [ebp-4]
push    edx
call    sub_401430
add     esp, 4
call    WSACleanup
xor     eax, eax
mov     esp, ebp
pop     ebp
retn
```

Avant de commencer toute connexion, le serveur convertit une IP en DWORD. L'IP est située en 40D1A4 mais elle est malheureusement cryptée :

0040D1A4h: 78 A6 10 86 74 9F 27 87 61 B5 FA 6E 6E A8 2C 9D ; x¦.†tŸ'‡aµúnn¨,▯

Il faut donc décrypter cette section : nous disposons déjà de la routine.

7. DECRYPTAGE DE LA SECTION DE DONNEES : FAILLE !

Revenons sur la routine de décryptage de la section de données :

décompression de la section code + décryptage de la section données

```
00422320  MOV EAX,forwarde.0040D000
00422325  MOV ECX,2FFC
0042232A  SUB DWORD PTR DS:[EAX+ECX],DEAD1337
00422331  ADD DWORD PTR DS:[EAX+ECX],ECX
00422334  XOR DWORD PTR DS:[EAX+ECX],EBX  <---------- EBX = 904DA2D6h
00422337  SUB ECX,4
0042233A  JNZ SHORT forwarde.0042232A
0042233C  ADD ESP,8
0042233F  POPAD
00422340  JMP forwarde.004223EE
```

J'ai indiqué ici la valeur nécessaire pour EBX. En fait, si vous regardez bien, il ne nous manque que cette valeur pour effectuer le décryptage. Pour la découvrir , il est inutile de bruteforcer quoique ce soit. Il suffit de chercher tranquillement les différentes adresses IP à trouver (constituées de valeurs choisies parmis les suivantes : 30h, 31h, 32h, 33h, 34h, 35h, 36h,

37h, 38h, 39h, 2Eh, 00h). On peut effectuer au préalable le SUB et le ADD pour obtenir une première phase de décryptage. Il ne nous reste alors plus qu'à déduire la valeur du EBX pour effectuer le XOR. On sait par exemple que les adresses IP sont stockées en tant que chaines de caractères, elles se terminent donc par 00h. On sait également qu'une adresse IP est composée obligatoirement de trois « . », dont le code héxa est 2Eh. On obtient donc EBX = 904DA2D6h. L'adresse IP située en 40D1A4h vaut donc :

172.31.194.182

Les pirates sont donc démasqués ☺

8 . REMERCIEMENTS

Je tiens à remercier **Nicolas Brulez** pour nous avoir proposé ce petit défi que j'ai apprécié. Je remercie également tout le STAFF de Securitech et notamment **Benjamin CAILLAT** pour avoir organisé ce challenge securitech 2005.

Un grand coucou à mes amis de route (particulièrement à Kaine), ceux du chan #uct, ceux du forum FC et évidemment ceux de la FRET.

BeatriX^FRET – Mardi 5 juillet 2005 -