# Analyse Malware

Malware K101.exe

Michael HOFMANN
Julie HOTTIER
Léo RENSON

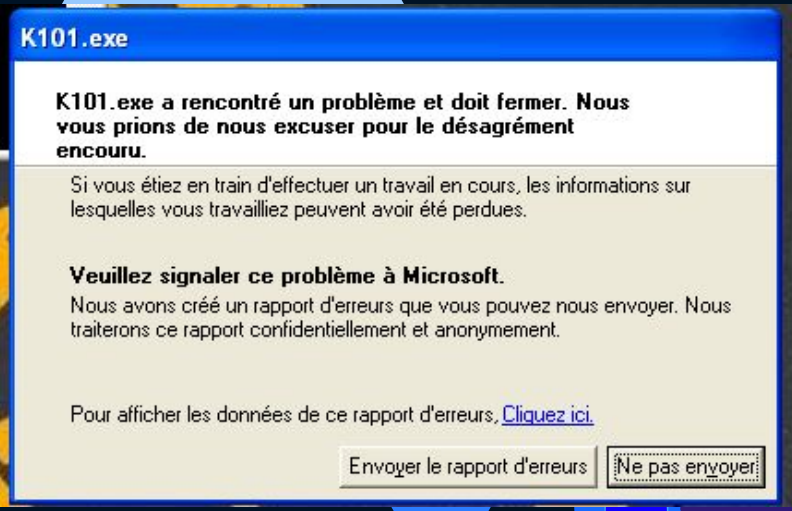
# Sommaire

# Analyse initiale

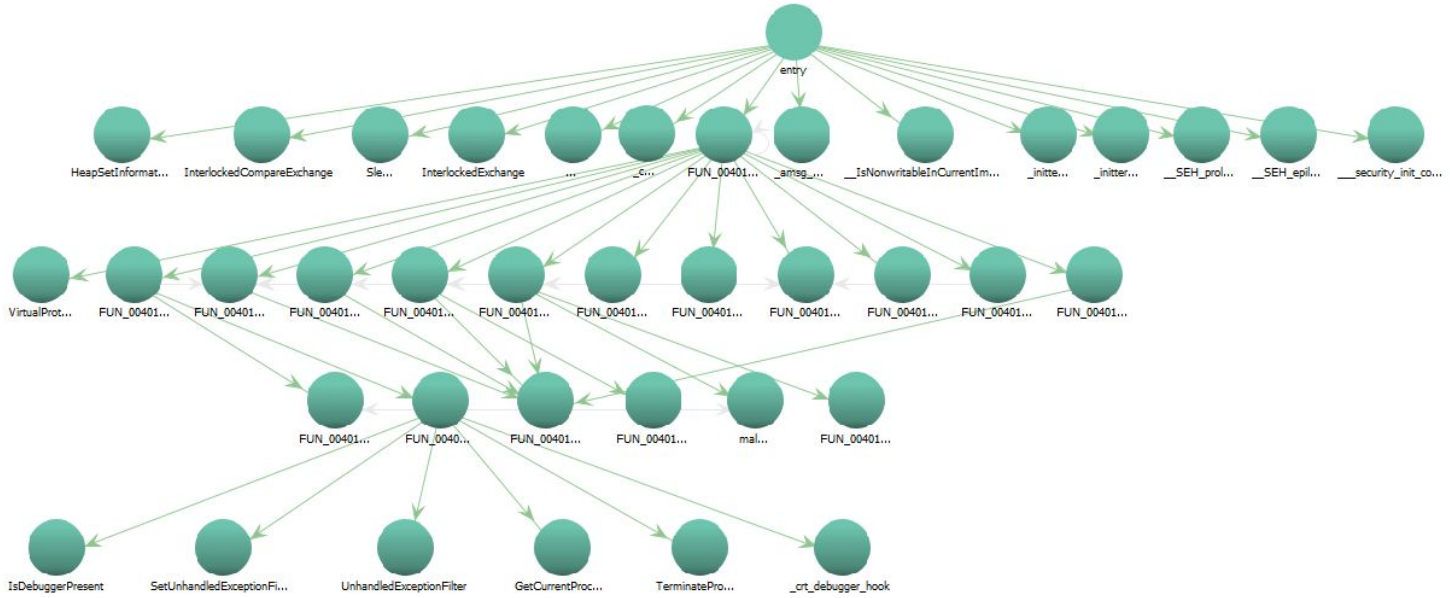
Premières opérations

# Premières exécutions

Exécutions simples pour observer le comportement du malware

# Différents appels de fonctions



# Informations en clair

| Address | Ordinal | Name | Library |
|---|---|---|---|
| 0000000000403000 | | VirtualProtect | KERNEL32 |
| 0000000000403004 | | GetCurrentProcessId | KERNEL32 |
| 0000000000403008 | | GetCurrentThreadId | KERNEL32 |
| 000000000040300C | | GetTickCount | KERNEL32 |
| 0000000000403010 | | QueryPerformanceCounter | KERNEL32 |
| 0000000000403014 | | DecodePointer | KERNEL32 |
| 0000000000403018 | | IsDebuggerPresent | KERNEL32 |
| 000000000040301C | | SetUnhandledExceptionFilter | KERNEL32 |
| 0000000000403020 | | UnhandledExceptionFilter | KERNEL32 |
| 0000000000403024 | | GetCurrentProcess | KERNEL32 |
| 0000000000403028 | | TerminateProcess | KERNEL32 |
| 000000000040302C | | EncodePointer | KERNEL32 |
| 0000000000403030 | | HeapSetInformation | KERNEL32 |
| 0000000000403034 | | InterlockedCompareExchange | KERNEL32 |
| 0000000000403038 | | Sleep | KERNEL32 |
| 000000000040303C | | InterlockedExchange | KERNEL32 |
| 0000000000403040 | | GetSystemTimeAsFileTime | KERNEL32 |
| 0000000000403048 | | _crt_debugger_hook | MSVCR100 |
| 000000000040304C | | ?terminate@@YAXXZ | MSVCR100 |
| 0000000000403050 | | _unlock | MSVCR100 |
| 0000000000403054 | | __set_app_type | MSVCR100 |
| 0000000000403058 | | _lock | MSVCR100 |
| 000000000040305C | | _onexit | MSVCR100 |
| 0000000000403060 | | _except_handler4_common | MSVCR100 |
| 0000000000403064 | | _invoke_watson | MSVCR100 |
| 0000000000403068 | | _controlfp_s | MSVCR100 |
| 000000000040306C | | _fmode | MSVCR100 |
| 0000000000403070 | | _commode | MSVCR100 |

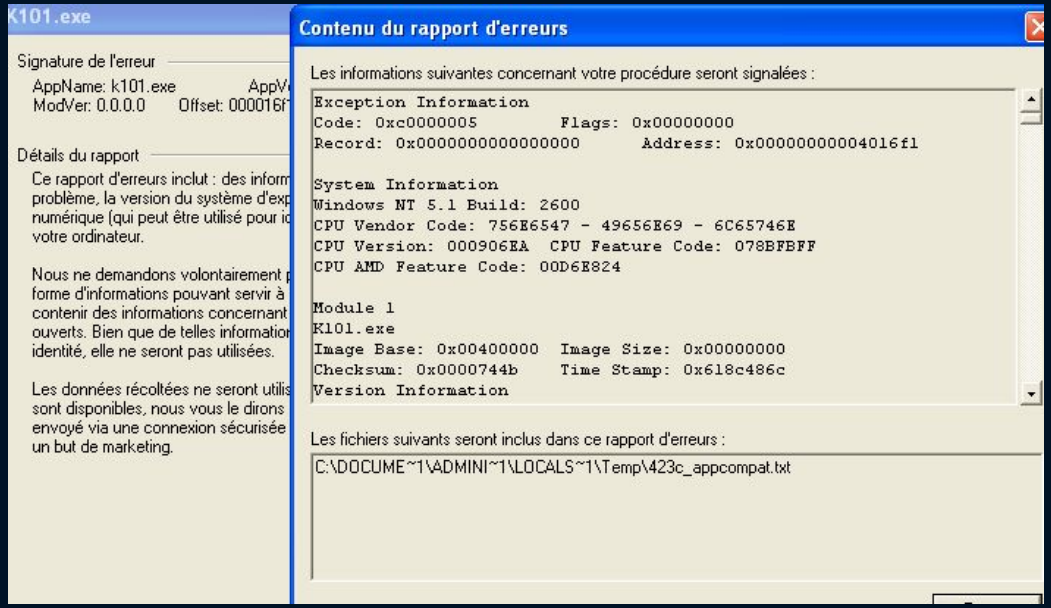
| | | |
|---|---|---|
| 0000000000403070 | _commode | MSVCR100 |
| 0000000000403074 | __setusermatherr | MSVCR100 |
| 0000000000403078 | _configthreadlocale | MSVCR100 |
| 000000000040307C | _initterm_e | MSVCR100 |
| 0000000000403080 | _initterm | MSVCR100 |
| 0000000000403084 | __initenv | MSVCR100 |
| 0000000000403088 | exit | MSVCR100 |
| 000000000040308C | _XcptFilter | MSVCR100 |
| 0000000000403090 | _exit | MSVCR100 |
| 0000000000403094 | _cexit | MSVCR100 |
| 0000000000403098 | __getmainargs | MSVCR100 |
| 000000000040309C | _amsg_exit | MSVCR100 |
| 00000000004030A0 | malloc | MSVCR100 |
| 00000000004030A4 | fscanf | MSVCR100 |
| 00000000004030A8 | _dllonexit | MSVCR100 |

| | |
|---|---|
| IMAGE_SECTION_HEADER | ".rsrc" |
| SectionFlags IMAGE_SCN_CNT_INITIALIZED_DATA \| IMAGE_SCN_MEM... | "@.reloc" |
| ?? 6Ch l | "la mere michelle a perdu son chat" |
| ?? 63h c | "courage" |
| ?? 6Fh o | "on t'aime" |
| ?? 62h b | "bg va" |


# En détail

Analyse en détail à partir du rapport d'erreurs

# Rapport d'erreurs




Adresse: 0x4016F1

# Fonction associée à l'adresse 0x4016F1

```
FUN_004016e0                                XREF[2]:     FUN_00401260:00401290(c),
                                                         FUN_00401d70:00401daf(c)
```

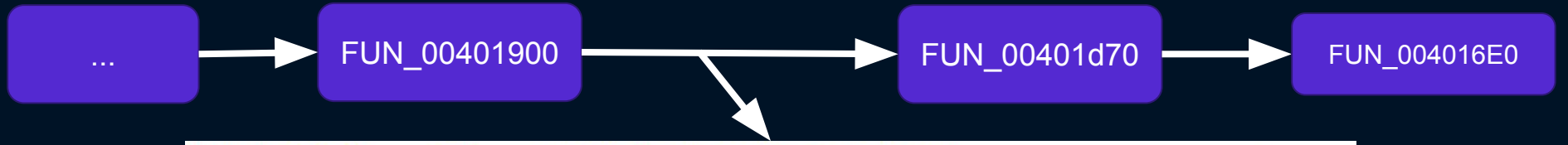
```
                    FUN_004016e0
004016e0 55             PUSH        EBP
004016e1 8b ec          MOV         EBP,ESP
004016e3 51             PUSH        ECX
004016e4 c7 45 fc       MOV         dword ptr [EBP + local_8],0x0
         00 00 00 00

                    LAB_004016eb
004016eb 8b 45 08       MOV         EAX,dword ptr [EBP + param_1]
004016ee 03 45 fc       ADD         EAX,dword ptr [EBP + local_8]
004016f1 0f be 08       MOVSX       ECX,byte ptr [EAX]
004016f4 85 c9          TEST        ECX,ECX
004016f6 74 0b          JZ          LAB_00401703
004016f8 8b 55 fc       MOV         EDX,dword ptr [EBP + local_8]
004016fb 83 c2 01       ADD         EDX,0x1
004016fe 89 55 fc       MOV         dword ptr [EBP + local_8],EDX
00401701 eb e8          JMP         LAB_004016eb
```

Désassemblage

```
1
2  int __cdecl FUN_004016e0(int param_1)
3
4  {
5    int local_8;
6
7    for (local_8 = 0; *(char *)(param_1 + local_8) != '\0'; local_8 = local_8 + 1) {
8    }
9    return local_8;
10 }
11
```

- Fonction qui utilise argv[1] (= param_1)
- Cette fonction réalise un calcul de taille d'une chaine de caractère

# Fonctions appelant 0x4016E0: FUN_00401D70

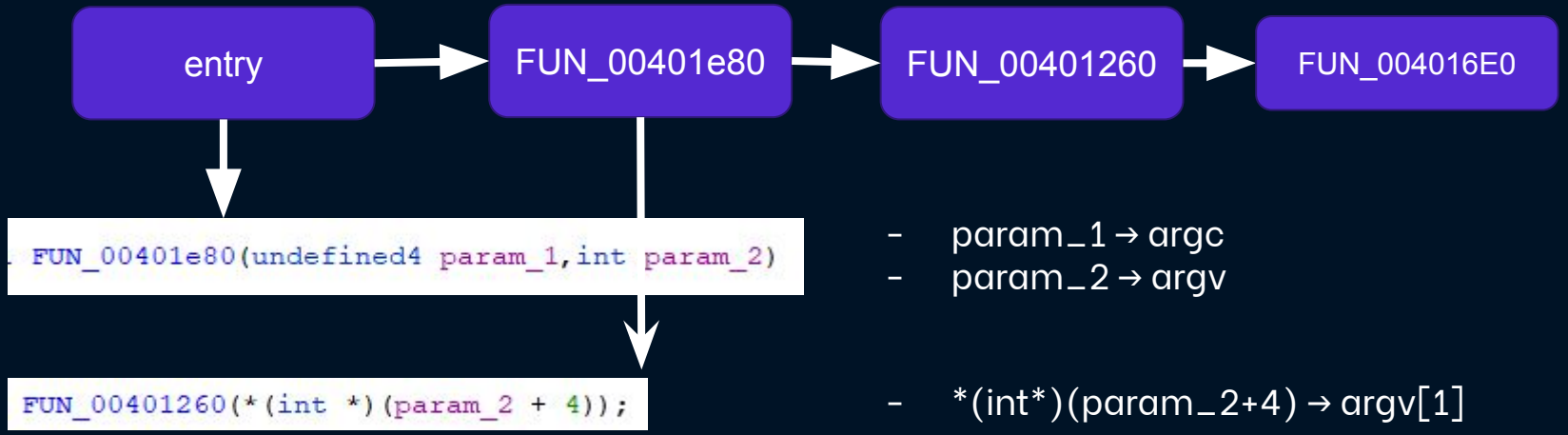



```
00401906 68 f4 30        PUSH       s_la_mere_michelle_a_perdu_son_cha_004030f4     = "la mere michelle a perdu son ...
         40 00
0040190b e8 60 04        CALL       FUN_00401d70                                    undefined4 * FUN_00401d70(int pa...
         00 00
```

```
FUN_00401d70((int)"la mere michelle a perdu son chat");
```

- N'utilise pas argv[1]
- Utilise un texte en clair

# Fonctions appelant 0x4016E0: FUN_00401260




- param_1 → argc
- param_2 → argv

- *(int*)(param_2+4) → argv[1]

# FUN_00401260

```
FUN_00401260(*(int *)(param_2 + 4));
```

- En paramètre : argv[1]

Variables intéressantes:
- int local_40[4] initialisé à 0
- int local_24 = FUN_004016E0(param_1) → len de argv[1]
- int Local_58: initialisé à 0 et incrémenté dans le while par 1 à chaque boucle

Deux opérations effectuées:
- Une boucle for pour peupler local_40
- Une boucle while avec un switch-case sur local_40[local_58]
  - Le switch case fait appel à différentes fonctions, les adresses sont obfusquées

# FUN_00401260 : peupler local_40

```
                03 00 00 00
004012a9 eb 09          JMP         LAB_004012b4

                    LAB_004012ab                                    XREF[1]:     0040130c(j)
004012ab 8b 4d b0       MOV         ECX,dword ptr [EBP + local_54]
004012ae 83 e9 01       SUB         ECX,0x1
004012b1 89 4d b0       MOV         dword ptr [EBP + local_54],ECX

                    LAB_004012b4                                    XREF[1]:     004012a9(j)
004012b4 83 7d b0 ff    CMP         dword ptr [EBP + local_54],-0x1
004012b8 7e 54          JLE         LAB_0040130e
004012ba 8b 4d b0       MOV         ECX,dword ptr [EBP + local_54]
004012bd 83 c1 01       ADD         ECX,0x1
004012c0 8b 45 e0       MOV         EAX,dword ptr [EBP + local_24]
004012c3 99             CDQ
004012c4 f7 f9          IDIV        ECX
004012c6 89 45 e4       MOV         dword ptr [EBP + local_20],EAX
004012c9 83 7d e4 0a    CMP         dword ptr [EBP + local_20],0xa
004012cd 7e 24          JLE         LAB_004012f3
004012cf 83 7d b0 00    CMP         dword ptr [EBP + local_54],0x0
004012d3 74 1e          JZ          LAB_004012f3
004012d5 8b 55 b0       MOV         EDX,dword ptr [EBP + local_54]
004012d8 c7 44 95       MOV         dword ptr [EBP + EDX*0x4 + -0x3c],0xa
         c4 0a 00
         00 00
004012e0 8b 45 b0       MOV         EAX,dword ptr [EBP + local_54]
004012e3 83 c0 01       ADD         EAX,0x1
004012e6 6b c0 0a       IMUL        EAX,EAX,0xa
004012e9 8b 4d e0       MOV         ECX,dword ptr [EBP + local_24]
```

```
for (local_54 = 3; local_54 < 0x80000000; local_54 = local_54 - 1) {
  local_20 = (int)local_24 / (int)(local_54 + 1);
  if ((local_20 < 0xb) || (local_54 == 0)) {
    local_40[local_54] = local_20;
    uVar1 = (int)local_24 % (int)(local_54 + 1);
    local_24 = uVar1;
  }
  else {
    local_40[local_54] = 10;
    local_24 = local_24 + (local_54 + 1) * -10;
    uVar1 = local_54;
  }
}
```

A chaque boucle :

- Local_20 ← LEN/i+1
- Si local_20 < 0xb = 11 → local_40[i] = local_20
- Sinon: local_40[i] = 10

Les valeurs de local_40 sont donc comprises entre 1 et 10

# FUN_00401260 : switch-case

```
                    switchD_00401351::switchD
00401351 ff 24 85       JMP         dword ptr [EAX*0x4 + ->switchD_00401351::caseD...= 00401358
         a8 16 40 00

                    switchD_00401351::caseD_1                       XREF[2]:     00401351(j), 004016a8(*
00401358 8b 4d ac       MOV         ECX,dword ptr [EBP + local_58]
0040135b 8b 44 8d c4    MOV         EAX,dword ptr [EBP + ECX*0x4 + -0x3c]
0040135f 69 c0 71       IMUL        EAX,EAX,0x395b71
         5b 39 00
00401365 99             CDQ
00401366 b9 45 72       MOV         ECX,0xa7245
         0a 00
0040136b f7 f9          IDIV        ECX
0040136d 8b 45 ac       MOV         EAX,dword ptr [EBP + local_58]
00401370 8b 4c 85 c4    MOV         ECX,dword ptr [EBP + EAX*0x4 + -0x3c]
00401374 83 c1 05       ADD         ECX,0x5
00401377 6b c9 0e       IMUL        ECX,ECX,0xe
0040137a 03 d1          ADD         EDX,ECX
0040137c 89 55 e8       MOV         dword ptr [EBP + local_1c],EDX
0040137f ba 7c 37       MOV         EDX,0x45377c
         45 00
00401384 33 55 e8       XOR         EDX,dword ptr [EBP + local_1c]
00401387 8d 45 08       LEA         EAX=>param_1,[EBP + 0x8]
0040138a 50             PUSH        EAX
0040138b 8b 4d ac       MOV         ECX,dword ptr [EBP + local_58]
0040138e 51             PUSH        ECX
0040138f ff d2          CALL        EDX
00401391 83 c4 08       ADD         ESP,0x8
00401394 e9 f9 02       JMP         LAB_00401692
```

```
while ((int)local_58 < 4) {
  switch(local_40[local_58]) {
  case 1:
    local_1c = (local_40[local_58] * 0x395b71) % 0xa7245 + (local_40[local_58] + 5) * 0xe;
    (*(code *)(local_1c ^ 0x45377c))(local_58,&param_1);
    break;
  case 2:
    local_2c = (local_40[local_58] * 0x394bd1) % 0xa7245 + (local_40[local_58] + 5) * 0xe;
    (*(code *)(local_2c ^ 0x4a36d2))(local_58,&param_1);
    break;
  case 3:
    uVar1 = (local_40[local_58] * 0x395b71) % 0xa71e1 + (local_40[local_58] + 5) * 0xe;
    (*(code *)(uVar1 ^ 0x44e333))(local_58,&param_1);
    FUN_00401bd0();
    (*(code *)(uVar1 ^ 0x44e3a3))(local_58,&param_1);
    break;
  case 4:
    local_c = (local_40[local_58] * 0x3c68b1) % 0xa7245 + (local_40[local_58] + 5) * 0xe;
    (*(code *)(local_c ^ 0x41460f))(local_58,&param_1);
    break;
  case 5:
    local_18 = (local_40[local_58] * 0x395b73) % 0xa7245 + (local_40[local_58] + 5) * 0xe;
    (*(code *)(local_18 ^ 0x44a534))(local_58,&param_1);
```

Switch sur les valeurs dans local_40:

- Chaque case appel une fonction obfusquée avec comme arguments local_58 et &param_1

# FUN_00401260 : switch-case

Exemple de fonction obfusquée, (code *) correspondant au typedef d'une fonction

```
case 1:
  local_1c = (local_40[local_58] * 0x395b71) % 0xa7245 + (local_40[local_58] + 5) * 0xe;
  (*(code *)(local_1c ^ 0x45377c))(local_58,&param_1);
  break;
```

Adresses des fonctions obfusquées:

- case 1: 0x401710
- case 2: 0x401780
- case 3: 0x401780
- case 4: 0x401900
- case 5: 0x401960
- case 6: 0x401A90
- case 7: 0x401B40
- case 8: 0x401900
- case 9: 0x401C10
- case 10: 0x401C70
- case 11: 0x401C60

Code en commun

FUN_00401000

```
0040173b eb 09          JMP         LAB_00401746

                    LAB_0040173d                                    XREF[1]:    0040176f(j)
0040173d 8b 4d f8       MOV         ECX,dword ptr [EBP + local_c]
00401740 83 c1 01       ADD         ECX,0x1
00401743 89 4d f8       MOV         dword ptr [EBP + local_c],ECX

                    LAB_00401746                                    XREF[1]:    0040173b(j)
00401746 8b 55 08       MOV         EDX,dword ptr [EBP + param_1]
00401749 83 c2 01       ADD         EDX,0x1
0040174c 39 55 f8       CMP         dword ptr [EBP + local_c],EDX
0040174f 7d 20          JGE         LAB_00401771
00401751 8b 45 0c       MOV         EAX,dword ptr [EBP + param_2]
00401754 8b 08          MOV         ECX,dword ptr [EAX]
00401756 0f b6 11       MOVSX       EDX,byte ptr [ECX]
00401759 52             PUSH        EDX
0040175a e8 a1 f8       CALL        FUN_00401000                    undefined4 f
         ff ff
0040175f 83 c4 04       ADD         ESP,0x4
00401762 8b 45 0c       MOV         EAX,dword ptr [EBP + param_2]
00401765 8b 08          MOV         ECX,dword ptr [EAX]
00401767 83 c1 01       ADD         ECX,0x1
0040176a 8b 55 0c       MOV         EDX,dword ptr [EBP + param_2]
0040176d 89 0a          MOV         dword ptr [EDX],ECX
0040176f eb cc          JMP         LAB_0040173d
```

```
for (local_8 = 0; local_8 < 1; local_8 = local_8 + 1) {
  if (param_1 != -1) {
    for (local_c = 0; local_c < param_1 + 1; local_c = local_c + 1) {
      FUN_00401000();
      *param_2 = *param_2 + 1;
    }
  }
}
```

# FUN_00401000

```
00401000 55             PUSH    EBP
00401001 8b ec          MOV     EBP,ESP
00401003 51             PUSH    ECX
00401004 53             PUSH    EBX
00401005 e8 26 00       CALL    FUN_00401030
         00 00
0040100a 89 45 fc       MOV     dword ptr [EBP + local_8],EAX
0040100d 0f be 45 08    MOVSX   EAX,byte ptr [EBP + Stack[0x4]]
00401011 85 c0          TEST    EAX,EAX
00401013 74 0f          JZ      LAB_00401024
00401015 58             POP     EAX
00401016 5b             POP     EBX
00401017 59             POP     ECX
00401018 8b 1b          MOV     EBX,dword ptr [EBX]
0040101a 53             PUSH    EBX
0040101b 51             PUSH    ECX
0040101c 53             PUSH    EBX
0040101d 50             PUSH    EAX
0040101e 83 ed 04       SUB     EBP,0x4
00401021 8b 45 fc       MOV     EAX,dword ptr [EBP + local_c]
```

```
undefined4 FUN_00401000(void)

{
  FUN_00401030();
  return 2;
}
```

Local_8: variable locale qui contient la valeur de retour de FUN_00401030

# FUN_00401030

Récupération d'une adresse d'une fonction à partir d'une autre adresse

```
bVar2 = true;
local_c = 0;
while (bVar2) {
  if (((((((char)fscanf_exref[local_c] * 0xff == -0x748b) &&
          ((char)fscanf_exref[local_c + 1] * 0xff == -0xff)) &&
         ((char)fscanf_exref[local_c + 2] * 0xff == 0x54ab)) &&
        (((char)fscanf_exref[local_c + 3] * 0xff == -0x748b &&
         ((char)fscanf_exref[local_c + 4] * 0xff == -0x13ec)))) &&
       (((char)fscanf_exref[local_c + 5] * 0xff == -0x728d &&
        (((char)fscanf_exref[local_c + 6] * 0xff == 0x44bb &&
         ((char)fscanf_exref[local_c + 7] * 0xff == 0xbf4)))))) &&
      ((char)fscanf_exref[local_c + 8] * 0xff == 0x4fb0)) &&
     (((((char)fscanf_exref[local_c + 9] * 0xff == 0x6996 &&
        ((char)fscanf_exref[local_c + 10] * 0xff == 0)) &&
       ((char)fscanf_exref[local_c + 0xb] * 0xff == -0xff)) &&
      ((((char)fscanf_exref[local_c + 0xc] * 0xff == 0x748b &&
        ((char)fscanf_exref[local_c + 0xd] * 0xff == 0x7f8)) &&
       (((char)fscanf_exref[local_c + 0xe] * 0xff == 0x6798 &&
        (((char)fscanf_exref[local_c + 0xf] * 0xff == -0x649b &&
         ((char)fscanf_exref[local_c + 0x10] * 0xff == -0x48b7))))))))))) {
    bVar2 = false;
  }
  else {
    local_c = local_c + 1;
  }
}
pcVar1 = fscanf_exref + local_c + -0x2286;
ppcVar3 = (code **)malloc(4);
*ppcVar3 = pcVar1;
```

# Anti-Debug

```
_DAT_0040414c = in_CS;
_DAT_00404158 = in_SS;
_DAT_00404088 = IsDebuggerPresent();
_crt_debugger_hook(1);
SetUnhandledExceptionFilter((LPTOP_LEVEL_EXCEPTION_FILTER)0x0);
UnhandledExceptionFilter((_EXCEPTION_POINTERS *)&PTR_DAT_004030ec);
if (_DAT_00404088 == 0) {
  _crt_debugger_hook(1);
}
uExitCode = 0xc0000409;
hProcess = GetCurrentProcess();
TerminateProcess(hProcess,uExitCode);
return;
```

# Conclusion

# Conclusion

- Comportement de “echo” avec le premier argument.
- Le programme plante s’il n’y a pas d’arguments.
- Il nous permet de reconnaître une adresse mémoire avec laquelle nous avons commencé notre analyse.
- On découvre de nombreuses fonctions qui s’appellent entre elles.
- Des fonctions obfusquées avec des calculs hexa ont été réalisés.
- Une fonction est plus obfusquée que les autres, utilisant des déplacements à partir d’une autre adresse mémoire.