

Reverse Engineering

Audric De Pierrepont, Hadrien Cortade, Gautier Bielik

Sommaire

- Exercice 1
 - Raisonnement
 - Réponses
- Exercice 2
 - Raisonnement
 - Réponses

Exercice 1 : raisonnement

- Analyse graphe IDA

```
nop
nop
nop
mov     edx, fs:30h
mov     al, [edx+2]
cmp     al, 0
jnz     short locret_401057
```

```
mov     eax, [edx+64h]
cmp     eax, 2
jbe     short locret_401057
```

```
call    $+5
pop     esi
lea     edi, (byte_4010EB - 401028h)[esi]
lea     esi, (word_4010F6 - 401028h)[esi]
nop
push    edi
call    sub_40110B
push    esi
call    sub_40110B
push    0
push    esi
push    edi
push    0
push    1361C78Eh
push    578D8483h
call    sub_4010FC
```

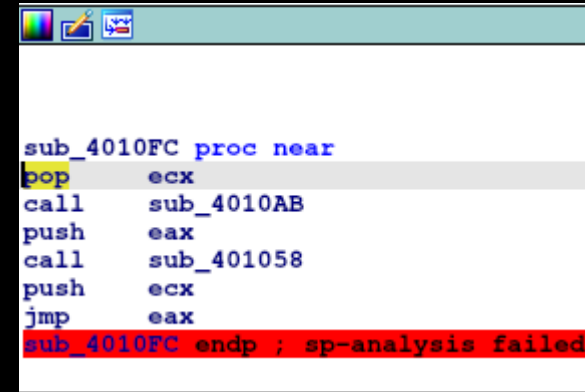
```
locret_401057:
retn
start endp
```

Exercice 1 : raisonnement

Min version	XP	XP SP2	2003/XP64	2003/XP64 SP1	Vista	Vista SP1	7	8 Pre RTM	8 Pre RTM	8.1 Update 1	10 TH2
Max version	XP SP1	XP SP3		2003/XP64 SP2		Vista SP2	7 SP1		8	10	
x86 offset offset:bitpos	Field Name										
0x0000	uint8_t InheritedAddressSpace										
0x0001	uint8_t ReadImageFileExecOptions										
0x0002	uint8_t BeingDebugged										
0x0003	uint8_t SpareBool			uint8_t BitField							
0x0003:0x00				uint8_t ImageUsesLargePages							
0x0003:0x01				uint8_t IsProtectedProcess							
0x0003:0x02				uint8_t IsLegacyProcess							uint8_t IsImageDynamicallyRelocated
0x0003:0x03				uint8_t IsImageDynamicallyRelocated							uint8_t SkipPatchingUser32Forwarders
0x0003:0x04				uint8_t SpareBits	uint8_t SkipPatchingUser32Forwarders						uint8_t IsPackagedProcess
0x0003:0x05					uint8_t SpareBits	uint8_t SpareBits				uint8_t IsPackagedProcess	uint8_t IsAppContainer
0x0003:0x06										uint8_t IsAppContainer	uint8_t IsProtectedProcessLight
0x0003:0x07										uint8_t SpareBits	
0x0004	void * Mutant										
0x0008	void * ImageBaseAddress										
0x000C	struct _PEB_LDR_DATA * Ldr										
0x0010	struct _RTL_USER_PROCESS_PARAMETERS * ProcessParameters										
0x0014	void * SubSystemData										
0x0018	void * ProcessHeap										
0x001C	struct _RTL_CRITICAL_SECTION * FastPebLock										
0x0020	void * FastPebLockRoutine		void * SparePtr1	void * AtlThunkSListPtr							
0x0024	void * FastPebUnlockRoutine		void * SparePtr2			void * IFEOKey					
0x0028					unsigned long						

Exercice 1 : raisonnement

- Deux fonctions :
 - hash vu en cours
 - hash plus compliquée



A screenshot of a debugger window showing assembly code. The code is for a function named `sub_4010FC` which is `proc near`. The instructions are: `pop ecx`, `call sub_4010AB`, `push eax`, `call sub_401058`, `push ecx`, and `jmp eax`. The final line, `sub_4010FC endp ; sp-analysis failed`, is highlighted in red. The `pop ecx` instruction is also highlighted in yellow.

```
sub_4010FC proc near
pop      ecx
call     sub_4010AB
push     eax
call     sub_401058
push     ecx
jmp      eax
sub_4010FC endp ; sp-analysis failed
```

Exercice 1 : raisonnement

- Données importantes :
 - « cclfxzvjit Code : »
 - 0x578D8483
 - 0X1361C78E

The screenshot shows a debugger window for 'output_002.exe - PID: 2944 - Module: output_002.exe - Thread: Main Thread 7296 - x32dbg'. The interface includes a menu bar, a toolbar, and several panes. The main pane displays assembly code with addresses, hex values, and mnemonics. The right pane shows the 'Masquer FPU' register window with values for EAX, EBX, ECX, EDI, EIP, EFLAGS, ZF, PF, AF, OF, SF, DF, CF, TF, IF, LastError, LastStatus, GS, FS, ES, DS, CS, and ST(0), ST(1). The bottom pane shows a memory dump with addresses, hex values, and ASCII characters. The status bar at the bottom indicates 'Paused' and 'INT3 breakpoint at output_002.0040111F'.

```
004010CE 81C7 33221100 add edi,112233
004010D4 01C7 add edi,eax
004010D6 EB E5 jmp output_002.40108D
004010D8 387CA4 24 cmp edi,dword ptr ss:[esp+24]
004010DC 8842 10 mov eax,dword ptr ds:[edx+10]
004010DF 8812 mov edx,dword ptr ds:[edx]
004010E1 75 D5 jne output_002.401088
004010E3 89424 1C mov dword ptr ss:[esp+1C],eax
004010E7 61 popad
004010E8 C2 0400 ret 4
004010EB 6363 6C arpl word ptr ds:[ebx+6C],sp
004010EE 66:78 7A js output_002.40116B
004010F1 76 6A jbe output_002.40115D
004010F3 697400 43 6F64653A imul esi,dword ptr ds:[eax+eax+43],3A65
004010F8 0059 E8 add byte ptr ds:[ecx-18],bl
004010FE A9 FFFFFFFF test eax,50FFFFFF
00401103 E8 50FFFFFF call output_002.401058
00401109 51 push ecx
0040110B 60 jmp eax
0040110D 887424 24 mov esi,dword ptr ss:[esp+24]
00401110 89F7 mov edi,esi
00401112 B2 8D mov dl,8D
00401114 AC lodsb
00401115 300D xor al,dl
00401117 FEC2 inc dl
00401119 AA stosb
0040111B 3C 00 cmp al,0
0040111D 75 F6 jne output_002.401114
0040111F 61 popad
00401121 C2 0400 ret 4
00401122 0000 add byte ptr ds:[eax],al
```

Registers (Masquer FPU):

EAX	00000004
EBX	0020D000
ECX	00401000 <output_002.EntryPoint>
EDI	0020D000
ESP	0060FF80
EIP	0060FF6C
ESI	004010F6 "Code:"
EDI	004010E8 "cclfxzvjit"
EIP	0040111F output_002.0040111F

Memory Dump:

Adresse	Hexa	ASCII
004010E8	63 6C 66 78 7A 76 6A 69 74 00 43 6F 64 65 3A	cclfxzvjit.Code:
004010F8	00 59 E8 A9 FF FF FF 50 FF FF 51 FF E0	.YesyyPepyoyya
00401108	60 88 74 24 24 89 F7 82 8D AC 30 D0 FE C2 AA 3C	.t\$.+*.u0pA*
00401118	00 75 F6 61 C2 04 00 00 00 00 00 00 00 00 00	.u0A.....
00401128	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401138	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401148	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401158	00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401168	00 00 00 00 00 00 00 00 00 00 00 00 00 00

Exercice 1 : raisonnement

- Problème encouru

The screenshot shows a debugger window for 'output_002.exe - PID: 7468 - Module: output_002.exe - Thread: Main Thread 3624 - x32dbg'. The interface includes a menu bar, a toolbar, and several panes. The 'CPU' pane shows the assembly code with the instruction pointer (EIP) at 004010BF. The 'Registers' pane on the right shows the values of various registers, including EAX (00000000), ECX (00401057), and EDI (00000000). The 'Stack' pane at the bottom shows the current stack frame, with the return address at 00401057. The 'Command' pane at the bottom shows the command 'mov eax, ebx' and the status 'Paused'.

Assembly code (EIP: 004010BF):

```
004010B0 31C0 xor eax, eax
004010B1 AC lods b
004010B2 46 inc esi
004010B3 85C0 test eax, eax
004010B4 74 13 jle output_002.004010B8
004010B5 3C 61 cmp al, 61
004010B6 7C 02 jnc output_002.004010C8
004010B7 2C 20 sub al, 20
004010B8 C1CF 14 ror edi, 14
004010B9 81C7 33221100 add edi, 33221100
004010BA 01C7 add edi, eax
004010BB EB E5 jmp output_002.004010BD
004010BC 3B7CA4 24 cmp edi, dword ptr ss:[esp+24]
004010BD 8842 10 mov eax, dword ptr ds:[edx+10]
004010BE 8B12 mov edx, dword ptr ds:[edx]
004010BF 75 D5 jne output_002.004010C8
004010C0 894424 1C mov dword ptr ss:[esp+1C], eax
004010C1 61 popad
004010C2 C2 0400 ret 4
004010C3 6363 6C arpl word ptr ds:[ebx+6C], sp
004010C4 66:78 7A js output_002.004010B8
004010C5 76 64 jbe output_002.004010B8
004010C6 697400 43 imul esi, dword ptr ds:[eax+eax+43], 3A65
004010C7 0059 E8 add byte ptr ds:[ecx-18], bl
004010C8 A9 FFFFFFFF test eax, 50FFFFFFF
004010C9 E8 50FFFFFF call output_002.00401058
004010CA 51 push ecx
004010CB 60 jmp eax
004010CC 8B7424 24 mov esi, dword ptr ss:[esp+24]
004010CD 89F7 mov edi, esi
004010CE B2 8D mov dl, 8D
```

Registers:

Register	Value
EAX	00000000
ECX	00401057
EDX	77955D94
EBP	0060FF80
ESP	0060FF38
ESI	00000000
EDI	00000000

Stack (ST(0) to ST(15)):

Index	Address	Value
0	00000000	00000000
1	00000000	00000000
2	00000000	00000000
3	00000000	00000000
4	00000000	00000000
5	00000000	00000000
6	00000000	00000000
7	00000000	00000000
8	00000000	00000000
9	00000000	00000000
10	00000000	00000000
11	00000000	00000000
12	00000000	00000000
13	00000000	00000000
14	00000000	00000000
15	00000000	00000000

Command: Les commandes sont séparées par des virgules (comme les instructions en assembleur) : mov eax, ebx

Status: Paused

First chance exception on 004010BF (C0000005, EXCEPTION_ACCESS_VIOLATION)

Temps perdu à déboguer: 0:03:58:32

Exercice 1 : réponses

1. Deux conditions :

le programme ne doit pas détecter qu'il est debuggé

- il faut minimum 2 processeurs

Exercice 1 : réponses

2. Structures :

PEB, LDR, TEB

- 3. Explication du shellcode

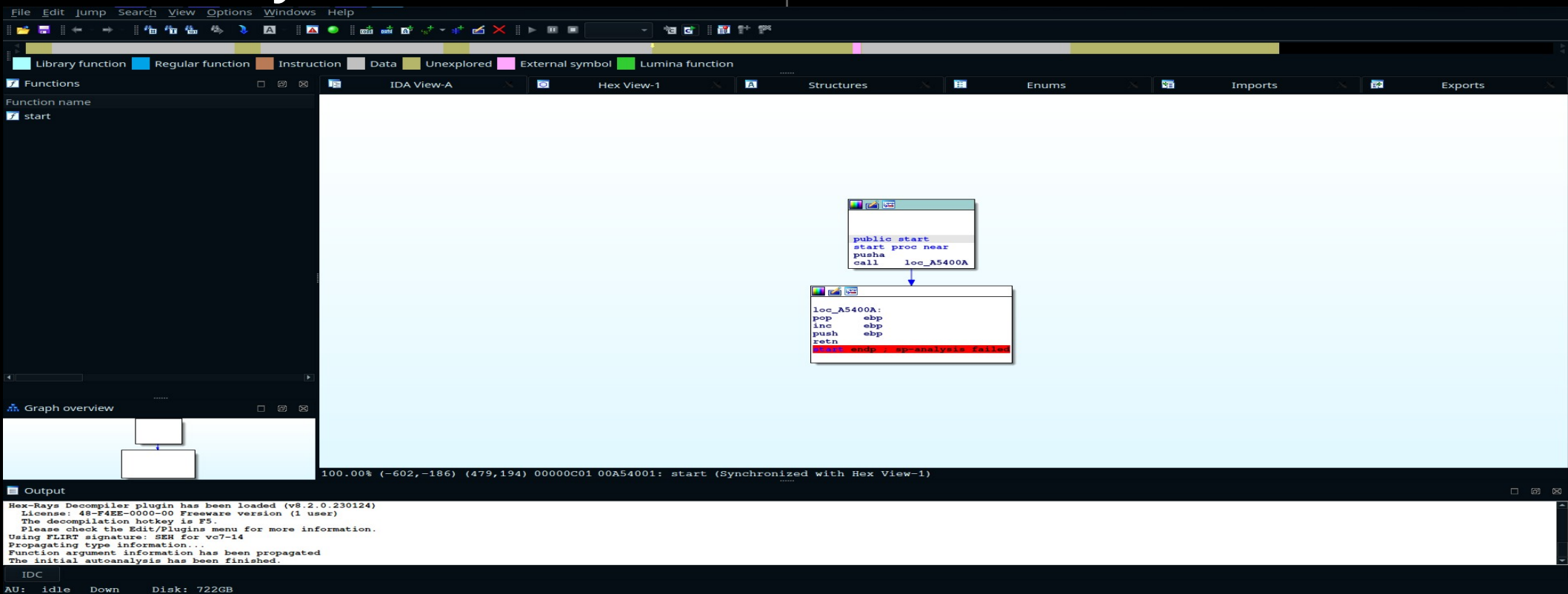
Exercice 1 : réponses

4. Hash de Kernel32.dll

Kernel32.DLL = D09F8780

Exercice 2 : raisonnement

- Analyse IDA



Exercice 2 : raisonnement

- Type de packing :

ASAPacking

Exercice 2 : raisonnement

Binaire dépaqué

The screenshot shows a debugger window for 'output_002.exe' (PID: 6736) running on a 32-bit system. The assembly view displays the following code:

```
00401000 nop
00401001 nop
00401002 call dword ptr ds:[<&GetTickCount>]
00401007 push 64
00401009 push 190
0040100B call dword ptr ds:[<&Beep>]
00401010 call dword ptr ds:[<&GetCommandLineA>]
00401015 mov edi, eax
00401017 push edi
00401019 call dword ptr ds:[<&strlen>]
0040101E mov ebx, eax
00401020 sub ebx, 5
00401023 mov ecx, edi
00401025 add ecx, ebx
00401028 mov eax, output_002.A51071
0040102B push eax
0040102D push ecx
0040102F call dword ptr ds:[<&strcmpA>]
00401034 cmp eax, 0
00401036 jne output_002.A51063
00401038 lea esi, dword ptr ds:[A51068]
0040103B lea edi, dword ptr ds:[A51064]
0040103E push 0
00401040 push esi
00401042 push edi
00401044 push 0
00401046 call dword ptr ds:[<&MessageBoxA>]
0040104B ret
0040104D inc edx
0040104F jb output_002.A510C8
00401051 jbe output_002.A510D8
00401053 and dword ptr ds:[eax], eax
```

The memory dump at the bottom shows the following data:

Adresse	Hexa
005FA2E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005FA2F0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005FA300	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005FA310	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005FA320	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005FA330	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005FA340	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005FA350	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005FA360	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005FA370	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

The command line at the bottom shows: 'output_002.exe: 00A51068 -> 00A51068 (0x00000001) bytes'.

Exercice 2 : réponses

Conditon :

Lancer le programme avec
« cclfx » comme argument

Merci à vous

