

MALWARES RETRO INGENIERIE DE CODE

Analyse d'un malware

- Kokou AMEZOTCHI
- Joseline YUEGO

Observations

Méthodes anti-debug

Cryptographie

Patch du Programme

Conclusion

Observations

- Le programme répond-t-il aux spécifications? Non
- Est-il malveillant? Potentiellement;
- Analyse, justification et proposition de solution.

Exécution

```
C:\Fichier Machine Ecran Ase
```

```
C:\> Invite de commandes de Visual Studio (2010) "C:\Documents and Settings\Administrateur\Bureau\nalware.exe" ... [A]  
Setting environment for using Microsoft Visual Studio 2010 x86 tools.  
C:\Program Files\Microsoft Visual Studio 10.0\VC>"C:\Documents and Settings\Administrateur\Bureau\nalware.exe" iertsgkj  
1°C  
C:\Program Files\Microsoft Visual Studio 10.0\VC>"C:\Documents and Settings\Administrateur\Bureau\nalware.exe" aczertgshlsdfrpl.sqdftryuiqsdf  
10°C  
C:\Program Files\Microsoft Visual Studio 10.0\VC>"C:\Documents and Settings\Administrateur\Bureau\nalware.exe" hqazexjii  
11°C  
C:\Program Files\Microsoft Visual Studio 10.0\VC>"C:\Documents and Settings\Administrateur\Bureau\nalware.exe" cdhqjpo4522222  
12°C  
C:\Program Files\Microsoft Visual Studio 10.0\VC>"C:\Documents and Settings\Administrateur\Bureau\nalware.exe" dsz  
13°C  
C:\Program Files\Microsoft Visual Studio 10.0\VC>"C:\Documents and Settings\Administrateur\Bureau\nalware.exe" edsz  
14°C  
C:\Program Files\Microsoft Visual Studio 10.0\VC>"C:\Documents and Settings\Administrateur\Bureau\nalware.exe" frtsqa  
15°C  
C:\Program Files\Microsoft Visual Studio 10.0\VC>"C:\Documents and Settings\Administrateur\Bureau\nalware.exe" 2frtsqa  
2°C  
C:\Program Files\Microsoft Visual Studio 10.0\VC>"C:\Documents and Settings\Administrateur\Bureau\nalware.exe" 3frtsqa  
3°C  
C:\Program Files\Microsoft Visual Studio 10.0\VC>"C:\Documents and Settings\Administrateur\Bureau\nalware.exe" qd75  
2°C  
C:\Program Files\Microsoft Visual Studio 10.0\VC>"C:\Documents and Settings\Administrateur\Bureau\nalware.exe" h  
2°C  
C:\Program Files\Microsoft Visual Studio 10.0\VC>  
C:\Program Files\Microsoft Visual Studio 10.0\VC>"C:\Documents and Settings\Administrateur\Bureau\nalware.exe" ?  
1°C  
C:\Program Files\Microsoft Visual Studio 10.0\VC>  
C:\Program Files\Microsoft Visual Studio 10.0\VC>  
C:\Program Files\Microsoft Visual Studio 10.0\VC>"C:\Documents and Settings\Administrateur\Bureau\nalware.exe" +c_e  
2°C  
C:\Program Files\Microsoft Visual Studio 10.0\VC>"C:\Documents and Settings\Administrateur\Bureau\nalware.exe"  
mchancet 
```

Figure 1: Exécution du code

```

-----
.text:00401000      push     ebp
.text:00401001      mov      ebp, esp
.text:00401003      mov      eax, [ebp+argc]
|.text:00401006      add      eax, 0FFFFFFEh
.text:00401009      jnz      short loc_401035
.text:0040100B      mov      edx, [ebp+argv]
.text:0040100E      mov      eax, [edx+4]
.text:00401011      lea      ecx, [ebp+argc]
.text:00401014      push     ecx
.text:00401015      push     offset Format ; "%X"
.text:0040101A      push     eax           ; Src
.text:0040101B      call     ds:scanf
.text:00401021      mov      ecx, [ebp+argc]
.text:00401024      push     ecx
.text:00401025      push     offset aD     ; "%d"
.text:0040102A      call     ds:printf
.text:00401030      add      esp, 14h
.text:00401033      jmp      short loc_401043
-----
.text:00401035      ;
.text:00401035      loc_401035:           ; CODE XREF: _main+91j
.text:00401035      push     offset aHchance ; "Hchance0"
00000040: 0000000000401006:  main+6

```

Figure 2: Code

Exécution du programme reçu

entrée : une chaîne de caractères de taille quelconque

comportement : extrait la première valeur, qui est ensuite interprétée comme un entier en format hexadécimal.

sortie :

- Si cette première valeur est un caractère hexadécimal `%X`, il renvoie la valeur décimale correspondante `"%d"` (entre **0** et **15**).
- Si cette première valeur n'est pas un caractère hexadécimal, il renvoie **2**.
- Pour tous les autres cas, il renvoie une chaîne de caractères : `"mÚchancetÚ"`.

Exécution : Contrôle de flux

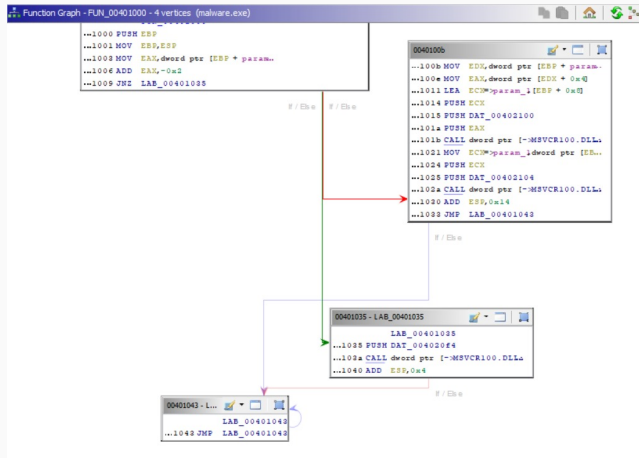
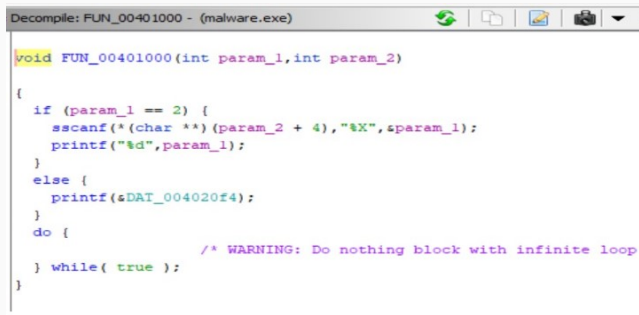


Figure 3: graphe du main

Code Approximatif



```
Decompile: FUN_00401000 - (malware.exe)

void FUN_00401000(int param_1,int param_2)
{
    if (param_1 == 2) {
        sscanf(*(char **) (param_2 + 4), "%X", &param_1);
        printf("%d", param_1);
    }
    else {
        printf(&DAT_004020f4);
    }
    do {
        /* WARNING: Do nothing block with infinite loop
    } while( true );
}
```

Figure 4: Main décompilé

Exécution : Remarques

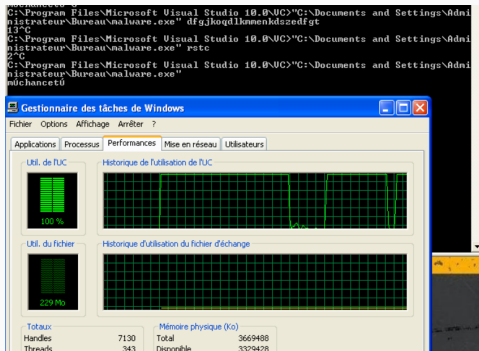


Figure 5: Comportement observé

- Message d'erreur volontairement ambigu : "mŭchancetŭ";
- Boucle infinie quelle que soit l'entrée (ou l'absence d'entrée);
- Saturation de l'UC qui reste à 100% dans tous les cas. ⚠ (Deni de service local?? ou maintien de l'interface graphique ouvert?)

Méthodes anti-debug

Méthodes anti-debug

IsDebuggerPresent : API Windows (Kernel32.dll)



Figure 6: Référence de l'anti debug

```
.text:00401387      mov     dword_40303C, 1
.text:00401391      mov     eax, __security_cookie
.text:00401396      mov     [ebp+var_328], eax
.text:0040139C      mov     eax, dword_403004
.text:004013A1      mov     [ebp+var_324], eax
.text:004013A7      call    ds:IsDebuggerPresent
.text:004013AD      mov     dword_403088, eax
.text:004013B2      push    1
```

Figure 7: Appel dans le programme

→ 004013A7

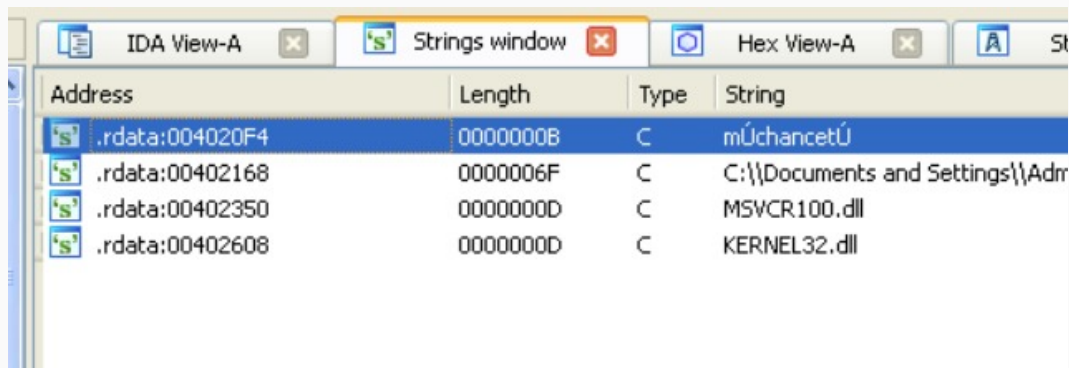
```
Python>import idaapi
DSTATE_RUN = 0
DSTATE_SUSP = 1
DSTATE_NOTASK = 2
def get_process_state():
    state = idaapi.get_process_state()
    if state == DSTATE_RUN:
        return "EN COURS D' EXECUTION"
    elif state == DSTATE_SUSP:
        return "SUSPENDU"
    elif state == DSTATE_NOTASK:
        return "TERMINER"
    else:
        return "ETAT INCONNU"
etat_processus = get_process_state()
print("Etat du processus : {}".format(etat_processus))
Etat du processus : SUSPENDU
```

Python |

Cryptographie

Observation

Nous n'avons pas observé d'éléments qui démontrerait une utilisation de procédés cryptographiques dans l'analyse de ce code à (payload / chaînes de caractères, ... chiffrées)



The screenshot shows the 'Strings window' in IDA Pro. The window has a title bar with 'Strings window' and a close button. Below the title bar is a table with four columns: 'Address', 'Length', 'Type', and 'String'. The table contains four entries, each with a small 's' icon in the first column. The first entry is selected and highlighted in blue.

Address	Length	Type	String
.rdata:004020F4	0000000B	C	mÚchancetÚ
.rdata:00402168	0000006F	C	C:\\Documents and Settings\\Adm
.rdata:00402350	0000000D	C	MSVCR100.dll
.rdata:00402608	0000000D	C	KERNEL32.dll

Figure 8: les strings

Patch du Programme

```
.text:00401387      mov     dword_40303C, 1
.text:00401391      mov     eax, __security_cookie
.text:00401396      mov     [ebp+var_328], eax
.text:0040139C      mov     eax, dword_403004
.text:004013A1      mov     [ebp+var_324], eax
.text:004013A7      call    ds:isDebuggerPresent
.text:004013AD      mov     dword_403088, eax
.text:004013B2      push    1
```

Figure 9: contourner isDebuggerPresent

isDebuggerPresent : annulation de l'appel

- remplacer le call par xor eax, eax
- forcer *dword_403088* à 0
- désactiver le security cookie

Annuler l'appel de isDebuggerPresent

Script

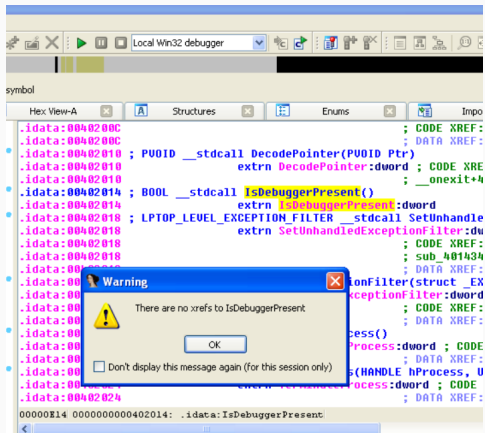
```
import idaapi
import idc

CALL_ISDEBUGGERPRESENT_ADDR = 0x004013A7
DW_RESULT_ADDR = 0x00403088

def patch_is_debugger_present():
    state = idaapi.get_process_state()
    patch_bytes = b"\x31\xC0\x90"
    idaapi.patch_bytes(CALL_ISDEBUGGERPRESENT_ADDR, patch_bytes)
    idaapi.patch_bytes(DW_RESULT_ADDR, b"\x00\x00\x00\x00")

patch_is_debugger_present()
```

Anti-Debug



Plus d'appel!

Figure 10: patch isDebuggerPresent

Patch boucle infinie?

```
.text:0040102A      call     ds:printf
.text:00401030      add     esp, 14h
.text:00401033      jmp     short loc_401043
.text:00401035      ; -----
.text:00401035      loc_401035:                                     ; CODE XREF: _main+9↑j
.text:00401035      push     offset aMschancets ; "mŭchancetŭ"
.text:00401035      call     ds:printf
.text:00401040      add     esp, 4
.text:00401043      loc_401043:                                     ; CODE XREF: _main+33↑j
.text:00401043      nop
```

Figure 11:

Ne rien faire

```
import idaapi

JMP_INFINITE_LOOP_ADDR = 0x00401043

def boucle():
    idaapi.patch_byte(JMP_INFINITE_LOOP_ADDR, 0x90) # NOP

boucle()
```

Conclusion

- **Fonctionnement du programme :**
 - Conversion hexadécimal vers décimal de la première lettre
 - Retourne 2 si l'entrée n'est pas un hexadécimal valide
 - Affiche une chaîne de caractères dans les autres cas
- **Problématiques identifiées :**
 - Consommation intensive d'UC : risque potentiel de déni de service
 - Présence d'un mécanisme anti-débogage
- **Solution apportée :**
 - Contournement réussi du mécanisme anti-débogage

Peut-être avons nous été aveugle?