



Malware

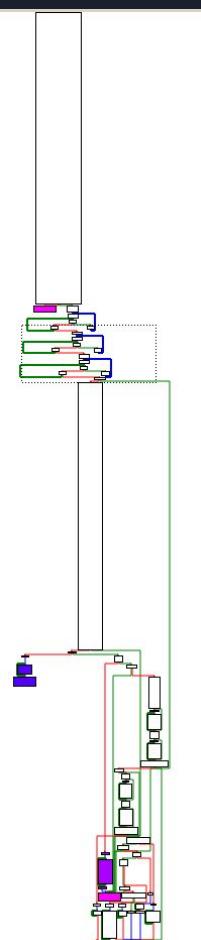
Groupe Thomas BIGEL, Christian
VASAUNE, Matthieu PERRIGOT

Malware 1

PESCE Teddy
MARTY Pierre
THIEMS Florian

Analyse du malware

Première exécution :



```
C:\WINDOWS\system32\cmd.exe
MARTY"
C:\Documents and Settings\Administrateur\Mes documents\Malware THIEMS_PESCE_MART
Y>Malware THIEMS_PESCE_MARTY.exe aaaaaaaaaaaaaaa
aaaaaaaaaaaaaaa
C:\Documents and Settings\Administrateur\Mes documents\Malware THIEMS_PESCE_MART
Y>_
```

```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrateur\Mes documents\Malware THIEMS_PESCE_MART
Y>Malware THIEMS_PESCE_MARTY.exe aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaa
La cle est trop longue
C:\Documents and Settings\Administrateur\Mes documents\Malware THIEMS_PESCE_MART
Y>_
```

Anti-debug

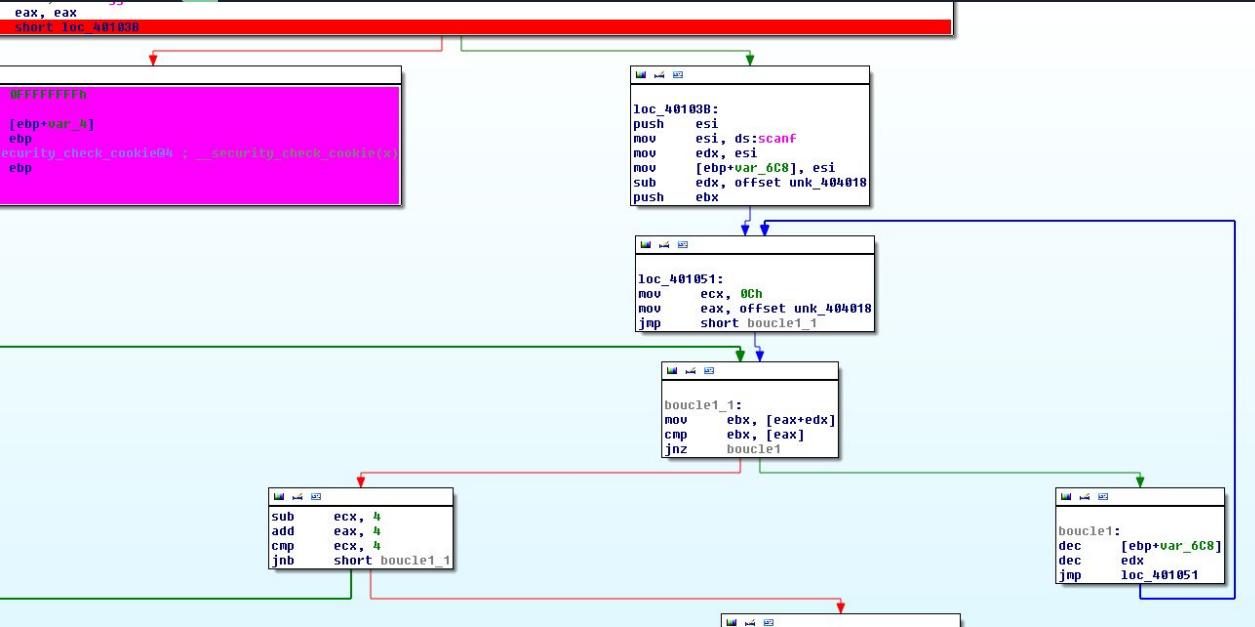
```
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

push    ebp
mov     ebp, esp
sub     esp, 60h
mov     eax, __security_cookie
xor     eax, ebp
mov     [ebp+var_4], eax
mov     eax, [ebp+argv]
push    edi
mov     edi, ds:IsDebuggerPresent
mov     [ebp+var_6CC], eax
call    edi ; IsDebuggerPresent
test   eax, eax
jz     short loc_A0103B
```

```
or     eax, 0FFFFFFFh
pop    edi
mov    ecx, [ebp+var_4]
xor    ecx, ebp
call    _security_check_cookie@4 ; _security_check_cookie@4
mov    esp, ebp
pop    ebp
ret
```

7 IsDebuggerPresent:
00401017
004010F6
00401DD3
00401F19
0040209D
004021C5
0040224D

Obfuscation



recherche pointeur fonction à partir de scanf:

[ebp+var_6C8]: printf

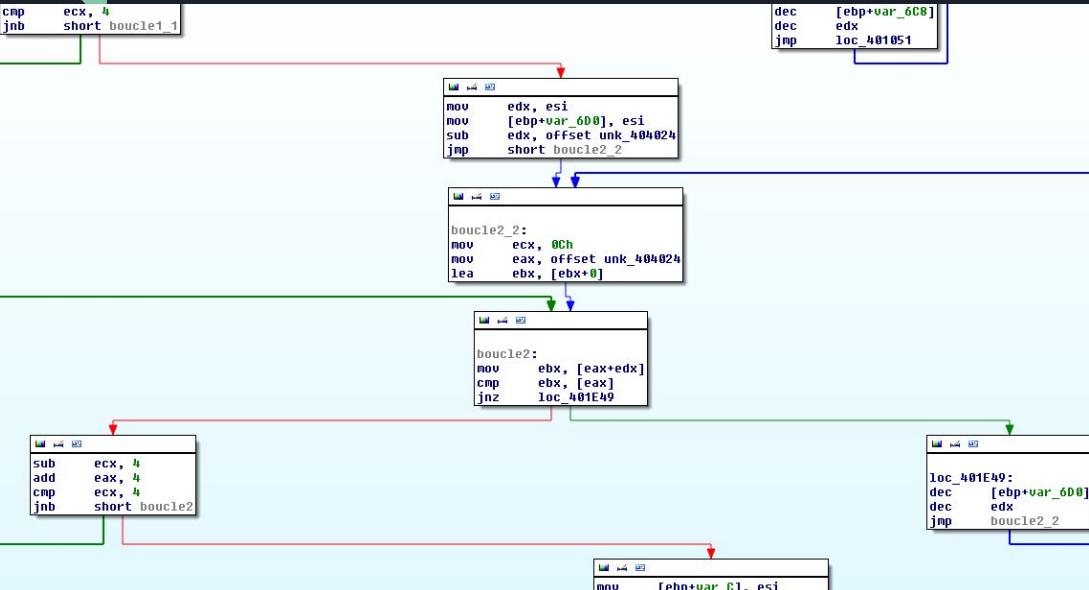
plages d'adresses:

0040103B à 0040105B

00401060 à 00401074

00401E3D à 00401E44

Obfuscation



recherche pointeur fonction à partir de scanf:

[ebp+var_6D0]: strcmp

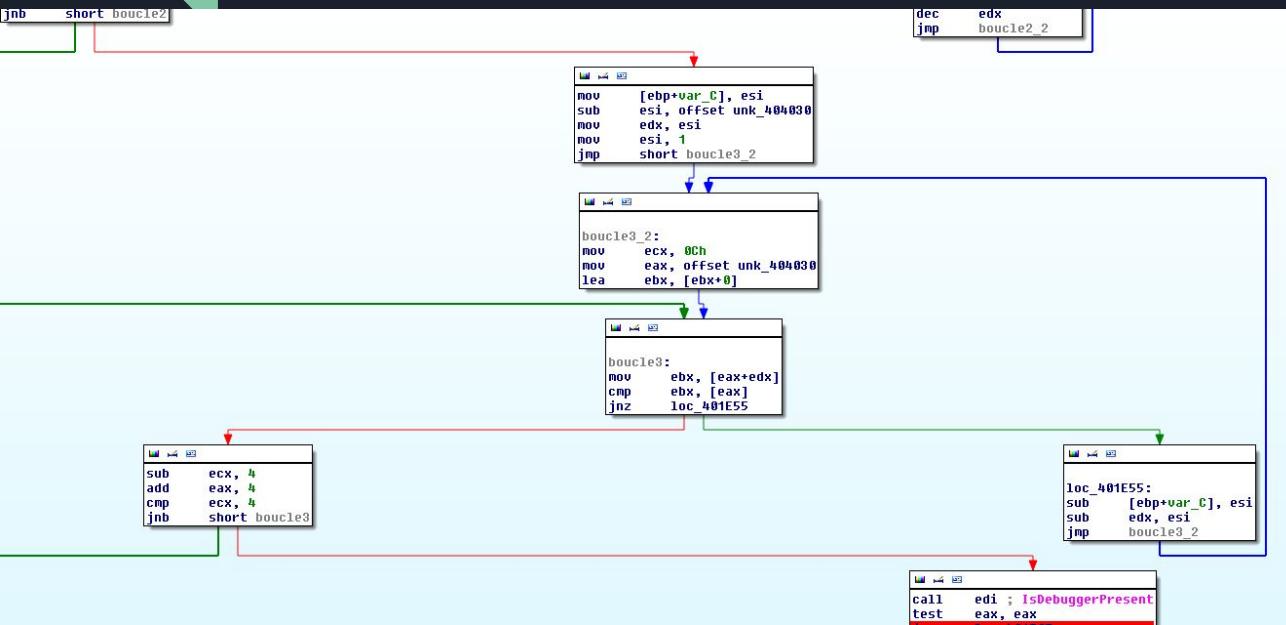
plages d'adresses:

00401076 à 00401084

00401E49 à 00401E50

00401090 à 004010B4

Obfuscation



recherche pointeur fonction à partir de scanf:

[ebp+var_C]: strlen

plages d'adresses:

004010B6 à 004010C6

00401E55 à 00401E5A

004010D0 à 004010F4

Utilisation de beaucoup de variables

- Utilisation de plein de variables négatives pour remonter à partir de ebp

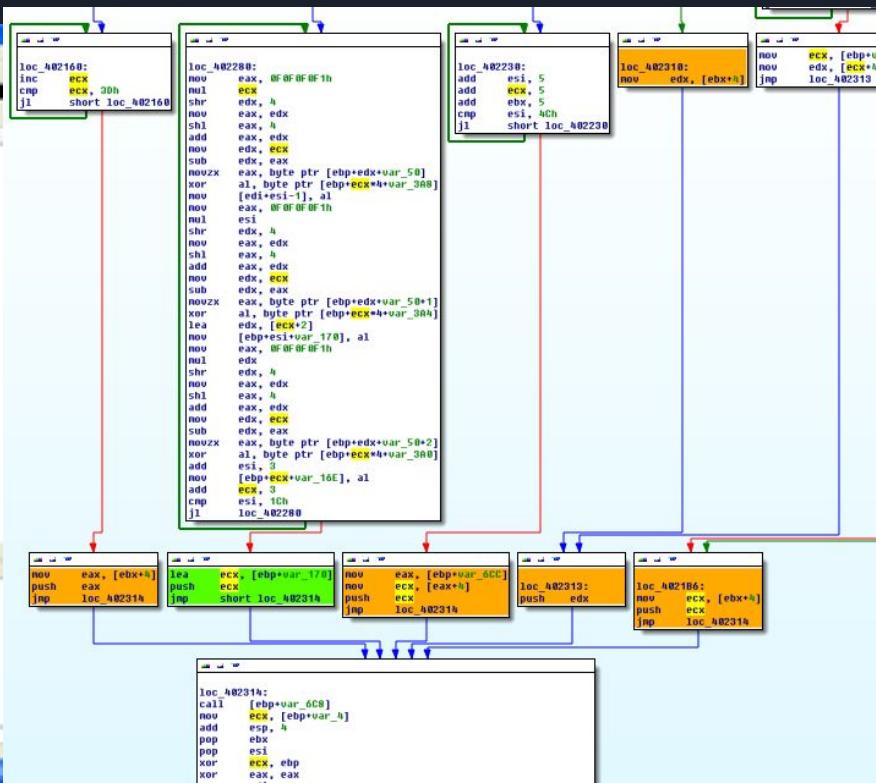
The screenshot shows the Immunity Debugger interface. On the left, the Function viewer lists various functions. In the center, the assembly view shows the following code:

```
loc_40103B:
push    esi
mov     esi, ds:scanf
mov     edx, esi
mov     [ebp+var_6C8], esi
sub    edx, offset unk_404018
push    ebx
```

Below the assembly, the Registers window shows:

Register	Value
esi	00000000
edx	00000000
ebx	00000000

The Registers window also displays memory values at addresses 404018 and 402314.



Chiffrement

```
push 3Fh          ; Size
push eax          ; Val
lea   edx, [ebp+Dst]
mov  bl, 30h
push edx          ; Dst
mov  [ebp+var_D8], 74201904h
mov  [ebp+var_D4], 21705113h
mov  [ebp+var_D0], 62441157h
mov  [ebp+var_CC], 78018696h
mov  [ebp+var_C8], 73795763h
mov  [ebp+var_18], 22549262h
mov  [ebp+var_14], 99442371h
mov  byte ptr [ebp+var_10], 47h
mov  [ebp+var_28], 40071410h
mov  [ebp+var_24], 56871223h
mov  [ebp+var_20], 206h
mov  [ebp+var_C], 15243110h
mov  [ebp+var_8], 36h
mov  [ebp+var_21C], bl
call memset
push 3Fh          ; Size
lea   eax, [ebp+var_108]
push 0             ; Val
push eax          ; Dst
mov  [ebp+var_10C], bl
call memset
push 3Fh          ; Size
lea   ecx, [ebp+var_298]
push 0             ; Val
push ecx          ; Dst
mov  [ebp+var_29C], bl
call memset
push 3Fh          ; Size
lea   edx, [ebp+var_258]
push 0             ; Val
push edx          ; Dst
mov  [ebp+var_25C], bl
call memset
mov  eax, [ebp+var_6CC]
mov  ecx, [eax+4]
push ecx
xor  esi, esi
call edi
add  esp, 34h
test eax, eax
jle  short loc_402036
```

Préparation des quatres chiffrements

4 memset

Plage d'adresses:
00401F23 à 00401FF3

Chiffrement

```
jne short loc_402030
mov    ebx, [ebp+var_6CC]
jmp    short loc_402000

loc_402000:
mov    ecx, [ebx+4]
mov    eax, 66666667h
imul   esi
sar    edx, 2
mov    eax, edx
shr    eax, 1Fh
add    eax, edx
lea    edx, [eax+eax*4]
add    edx, edx
mov    eax, esi
sub    eax, edx
mov    dl, byte ptr [ebp+eax+var_28]
xor    dl, [esi+ecx]
push   ecx
mov    [ebp+esi+var_1DC], dl
inc    esi
call   edi
add    esp, 4
cmp    esi, eax
jl    short loc_402000
```

Chiffrement de l'entrée

plage d'adresses:

00401FF5 à 00401FFB

00402000 à 00402034

plage adresse liste utilisé:

000000000012FF54 à 000000000012FF5D

{10h, 14h, 7, 40h, 23h, 12h, 87h, 56h, 6, 2}

```
loc_402036:
mov    ebx, [ebp+eax+var_18]
mov    eax, [ebx+4]
push   eax
xor    esi, esi
call   edi
add    esp, 4
test   eax, eax
jle    short loc_402083

loc_402050:
mov    ecx, [ebp+4]
mov    eax, 38E38E39h
imul  esi
sar    edx, 1
mov    eax, edx
shr    eax, 1Fh
add    eax, edx
lea    edx, [eax+eax*8]
mov    eax, esi
sub    eax, edx
mov    dl, byte ptr [ebp+eax+var_18]
xor    dl, [esi+ecx]
push   ecx
mov    [ebp+esi+var_29C], dl
inc    esi
call   edi
add    esp, 4
cmp    esi, eax
jle    short loc_402050
```

Chiffrement

Chiffrement de l'entrée
plage d'adresses:
00402050 à 00402081

plage adresse liste utilisé:
000000000012FF64 à 000000000012FF6C
{62h, 92h, 54h, 22h, 71h, 23h, 44h, 99h, 47h}

Chiffrement

The screenshot shows a debugger interface with three assembly code snippets:

- Top snippet:** `st eax, eax` at address `loc_401F05`. A red box highlights this line.
- Middle snippet:** `mov edx, [ebx+4]`, `push edx`, `xor esi, esi`, `call edi`, `add esp, 4`, `test eax, eax`, `jle short loc_4020F3`.
- Bottom snippet:** `lea ebx, [ebx+0]`.
- Bottom-most snippet:** `loc_4020C0:` `mov ecx, [ebx+4]`, `mov eax, 66666667h`, `imul esi`, `sar edx, 1`, `mov eax, edx`, `shr eax, 1Fh`, `add eax, edx`, `lea eax, [eax+eax*4]`, `mov edx, esi`, `sub edx, eax`, `mov al, byte ptr [ebp+edx+var_0]`, `xor al, [esi+ecx]`, `push ecx`, `mov [ebp+esi+var_21C], al`, `inc esi`, `call edi`, `add esp, 4`, `cmp esi, eax`, `jle short loc_4020C0`.

Arrows indicate the flow of control between these snippets. Red arrows point from the top snippet to the middle one, and from the middle one to the bottom-most snippet. A green arrow points from the bottom-most snippet back up to the middle one.

Chiffrement de l'entrée
plage d'adresses:
004020C0 à 004020F1

plage adresse liste utilisé:
000000000012FF70 à 000000000012FF74
{10h, 31h, 24h, 15h, 36h}

Chiffrement

```
loc_4020F3:
mov    ecx, [ebx+4]
push   ecx
xor    esi, esi
call   edi
add    esp, 4
test   eax, eax
jle    short loc_402100

loc_402102:
mov    ecx, [ebx+4]
mov    eax, 66666667h
imul  esi
sar    edx, 3
mov    eax, edx
shr    eax, 1Fh
add    eax, edx
lea    edx, [eax+eax*4]
add    edx, edx
add    edx, edx
mov    eax, esi
sub    eax, edx
mov    dl, byte ptr [ebp+eax+var_10]
xor    dl, [esi+ecx]
push   ecx
mov    [ebp+esi+var_25C], dl
inc    esi
call   edi
add    esp, 4
cmp    esi, eax
jle    short loc_402102
```

Chiffrement de l'entrée
plage d'adresses:
00402102 à 0040213B

plage adresse liste utilisé:
000000000012FEA4 à 000000000012FEB7
{4, 19h, 20h, 74h, 13h, 51h, 70h, 21h, 57h, 11h, 44h, 62h,
96h, 86h, 1, 78h, 63h, 57h, 79h, 73}



Outils

IDA: graphes, analyse des instructions

Debugger de IDA

Python: reproduction de l'algo de chiffrement -> algo de déchiffrement

Utilisation de exrex

Outils: python

chiffrement.py

```
def main():
    listeChiffrement = ['0x10', '0x14', '0x7', '0x40', '0x23', '0x12',
'0x87', '0x56', '0x6', '2']
    i = 0
    cle = ''
    for char in 'q&e%Ft\`0atvauasò3`0#%6qBpÙ3g1$':
        print(hex(int(listeChiffrement[i], 16) ^ int(hex(ord(char)), 16)))
        cle += chr(int(listeChiffrement[i], 16) ^ int(hex(ord(char)), 16))
        i += 1
        if i >= len(listeChiffrement):
            i = 0
    print(cle)
if __name__ == '__main__':
    main()
```

```
import subprocess
import exrex

def main():
    while 1:
        key = exrex.getone('a2beef.66cdbf5Ba.ef23111ab.ea346')
        output = subprocess.check_output(['C:\Documents and
Settings\Administrateur\Mes
documents\Malware _THIEMS _PESCE _MARTY\Malware _THIEMS _PESCE _MARTY.exe ',
key])
        if output != key:
            print(key + ' ' + output)

if __name__ == '__main__':
    main()
```

brute.py et brutePrec.py

```
key = exrex.getone('a2beef[3-7]66cdbf5.adef23111abfea346')
```

Trouver la clé secrète

Trouver le bon strcmp

00402257 à
0040226A

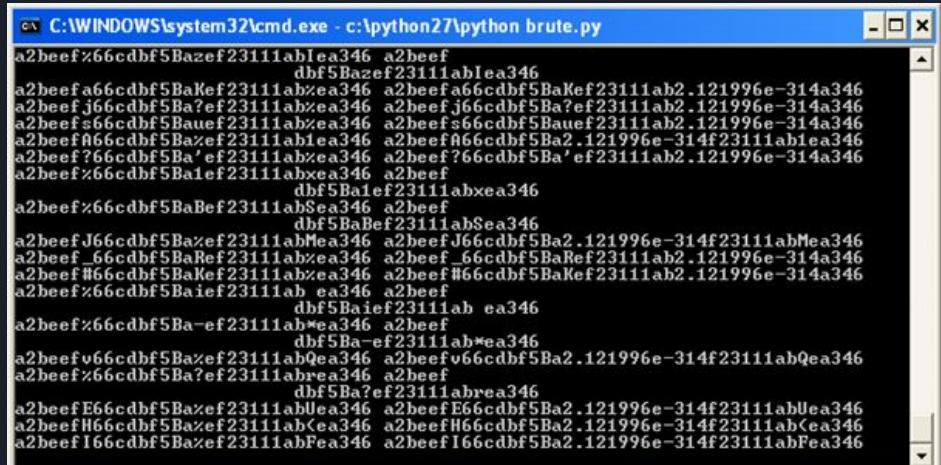
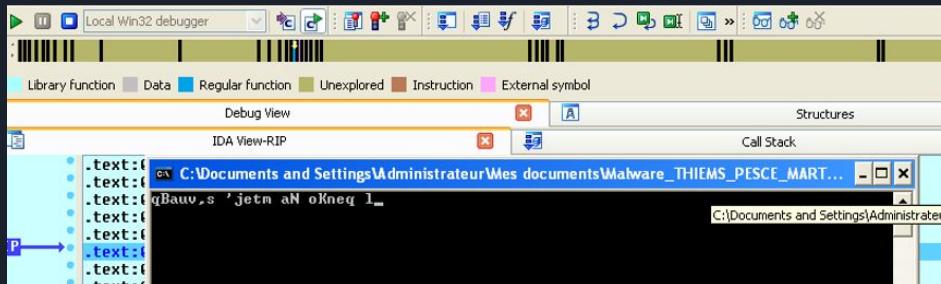
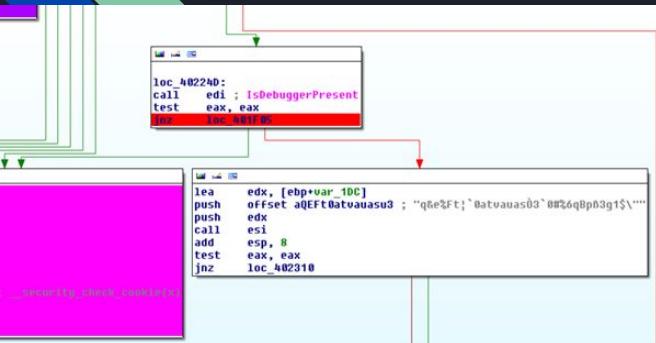
bon hash: q&e%Ft\`OatvauasÒ3`0#%6qBpß3g1\$"

1ere methode:

a2beef!66cdbf5BaUef23111abXea346

a2beefX66cdbf5BaXef23111abXea346

brute force: a2beef666cdbf5Badef23111abfea346



Trouver la clé secrète

bon hash: q&e%Ft!`0atvauasÓ3`0#%6qBpß3g1\$"

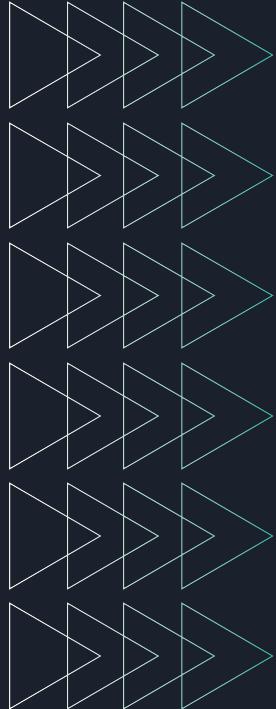
2eme methode:

brute force: a2beef666cdbf5Badef23111abfea346

```
C:\WINDOWS\system32\cmd.exe
V>Malware_THIEMS_PESCE_MARTY.exe a2beef766cdbf5fadef23111abfea346
a2beef766cdbf5fadef23111abfea346
C:\Documents and Settings\Administrateur\Mes documents\Malware_THIEMS_PESCE_MART
V>Malware_THIEMS_PESCE_MART.exe "a2beef766cdbf5 adef23111abfea346"
a2beef766cdbf5 adef23111abfea346
C:\Documents and Settings\Administrateur\Mes documents\Malware_THIEMS_PESCE_MART
V>Malware_THIEMS_PESCE_MART.exe a2beef366cdbf5Badef23111abfea346
a2beef366cdbf5Badef23111abfea346
C:\Documents and Settings\Administrateur\Mes documents\Malware_THIEMS_PESCE_MART
V>Malware_THIEMS_PESCE_MART.exe a2beef366cdbf5Badef23111abfea346
a2beef366cdbf5Badef23111abfea346
C:\Documents and Settings\Administrateur\Mes documents\Malware_THIEMS_PESCE_MART
V>Malware_THIEMS_PESCE_MART.exe a2beef466cdbf5Badef23111abfea346
a2beef466cdbf5Badef23111abfea346
C:\Documents and Settings\Administrateur\Mes documents\Malware_THIEMS_PESCE_MART
V>Malware_THIEMS_PESCE_MART.exe a2beef566cdbf5Badef23111abfea346
a2beef566cdbf5Badef23111abfea346
C:\Documents and Settings\Administrateur\Mes documents\Malware_THIEMS_PESCE_MART
V>Malware_THIEMS_PESCE_MART.exe a2beef666cdbf5Badef23111abfea346
Bravo, c'est la bonne cle !
C:\Documents and Settings\Administrateur\Mes documents\Malware_THIEMS_PESCE_MART
V>Malware_THIEMS_PESCE_MART.exe a2beef766cdbf5Badef23111abfea346
a2beef766cdbf5Badef23111abfea346
C:\Documents and Settings\Administrateur\Mes documents\Malware_THIEMS_PESCE_MART
V>
```



Résumé analyse du malware 1



Anti-debug - utilisation de beaucoup de IsDebuggerPresent() tout le long du malware

Obfuscation de fonction en partant de scanf pour les fonctions printf, strcmp et strlen

Utilisation de plein de **variables négatives** pour remonter à partir de ebp

Utilisation de **leurre**s avec strcmp

Beaucoup de mov ou d'instructions inutiles pour essayer de **brouiller** l'analyse

Chiffrement faible caractère à caractère

Malware 2

TENEDE TENE Miguel Bryan
DUVERNIER Alban
PORCU Baptiste

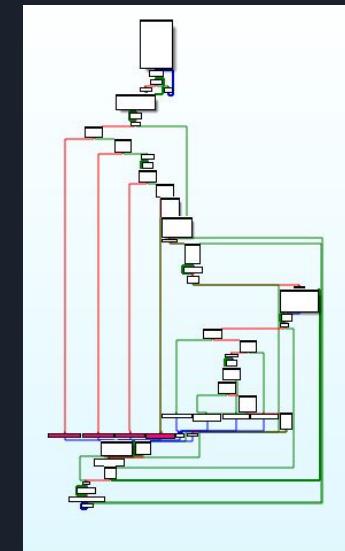
Analyse du malware

Première exécution :

```
C:\Documents and Settings\Administrateur\Bureau\share\malware\malware.exe
Entrez un message de moins de 64 lettres et écrit en hexa.
27897983ABC28
Le mot entré est : 27897983ABC28
Son hash en MD5 vaut: 55a77d19fb84c6c99853fb373a3c685c

Entrez un message de moins de 64 lettres et écrit en hexa.
387098BACD762
Son hash en MD5 vaut: ab267becce731cf5d1b337655244fec2

Entrez un message de moins de 64 lettres et écrit en hexa.
```



Malware sur IDA

Anti-debug

```
loc_4011CB:  
lea    edx, [ebp+pbData]  
push   edx  
push   offset aLeNotEntrsEstS ; "Le mot entré est : %s\n"  
call   esi  
mov    ebx, [ebp+Size]  
push   edi  
call   sub_401000  
push   offset Dest  
push   offset aSonHashEnMd5Va ; "Son hash en MD5 vaut: %s\n"  
call   esi  
add   esp, 14h  
mov    eax, large fs:30h  
mov    [ebp+var_90], eax  
mov    eax, [ebp+var_90]  
cmp    byte ptr [eax+2], 1  
jz     loc_401480
```

.text:004011F1

mov eax, large fs:30h

.text:0040120D

test byte ptr [eax+68h], 70h
jnz loc_401480

.text:00401211

```
mov    [ebp+var_90], eax  
mov    eax, [ebp+var_90]  
cmp    byte ptr [eax+2], 1  
jz     loc_401480
```

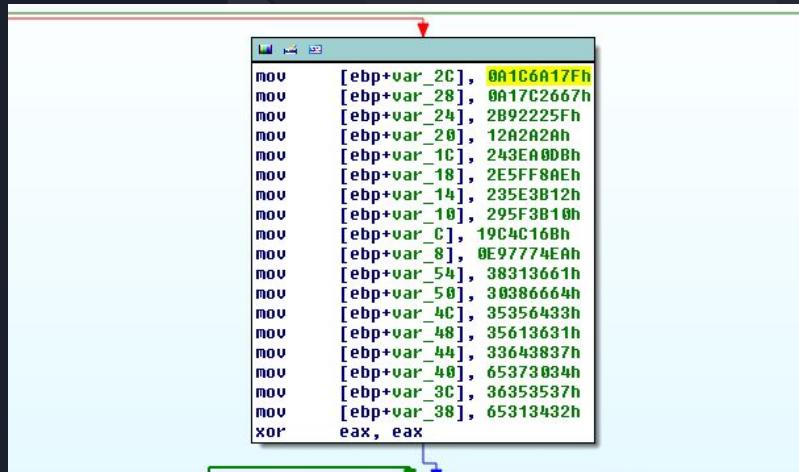
```
test   byte ptr [eax+68h], 70h  
jnz   loc_401480
```

Utilisation de variables

```
; Attributes: bp-based frame
; int _cdecl main(int argc, const char **argv, const char **envp)
_main proc near

var_94= dword ptr -94h
var_90= dword ptr -90h
var_8C= dword ptr -8Ch
hProv= dword ptr -88h
Size= dword ptr -84h
hHash= dword ptr -80h
pbData= byte ptr -7Ch
var_54= dword ptr -54h
var_50= dword ptr -50h
var_4C= dword ptr -4Ch
var_48= dword ptr -48h
var_44= dword ptr -44h
var_40= dword ptr -40h
var_3C= dword ptr -3Ch
var_38= dword ptr -38h
var_2C= dword ptr -2Ch
var_28= dword ptr -28h
var_24= dword ptr -24h
var_20= dword ptr -20h
var_1C= dword ptr -1Ch
var_18= dword ptr -18h
var_14= dword ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

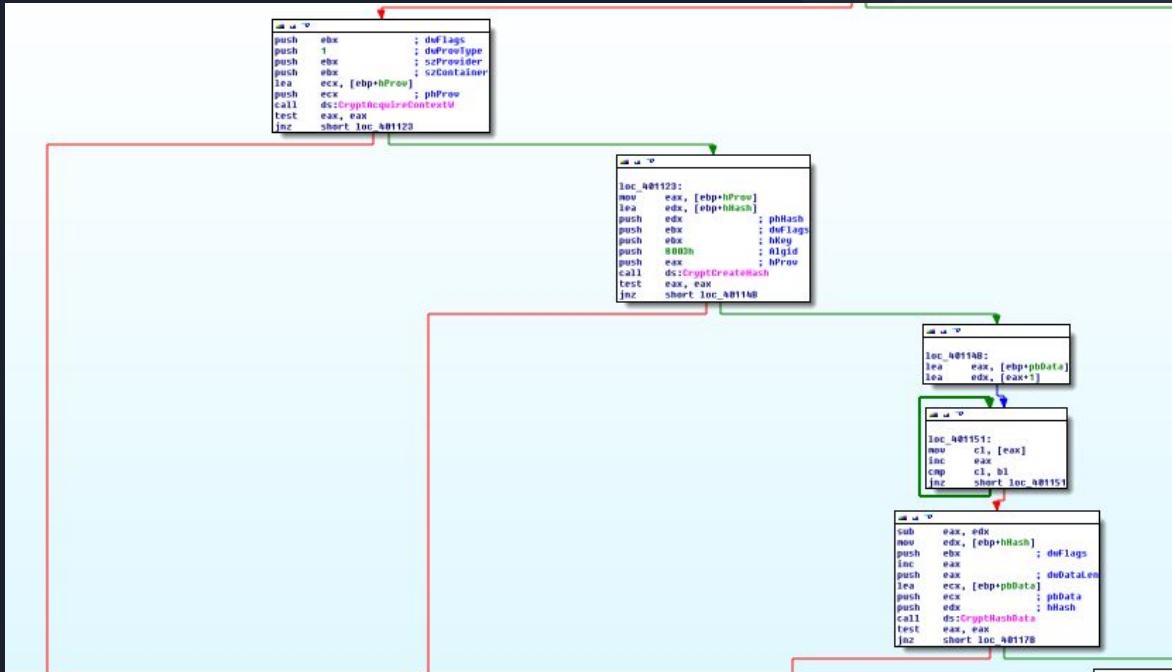
push    ebp
mov     ebp, esp
sub    esp, 94h
```



The screenshot shows a debugger interface with assembly code. The code is annotated with variable names in green, such as var_2C, var_28, var_24, etc., which correspond to the local variables defined in the main function. The assembly code includes instructions like mov [ebp+var_2C], 0A1C6A17Fh and xor eax, eax.

```
mov    [ebp+var_2C], 0A1C6A17Fh
mov    [ebp+var_28], 0A17C2667h
mov    [ebp+var_24], 2B92225Fh
mov    [ebp+var_20], 12A292Ah
mov    [ebp+var_1C], 243EA0D8h
mov    [ebp+var_18], 2E5FF8AEh
mov    [ebp+var_14], 235E3B12h
mov    [ebp+var_10], 295F3B10h
mov    [ebp+var_C], 19C4C16Bh
mov    [ebp+var_8], 0E97774EAh
mov    [ebp+var_54], 38313661h
mov    [ebp+var_50], 30386664h
mov    [ebp+var_4C], 35356433h
mov    [ebp+var_48], 35613631h
mov    [ebp+var_44], 33643837h
mov    [ebp+var_40], 65373034h
mov    [ebp+var_3C], 36353537h
mov    [ebp+var_38], 65313432h
xor    eax, eax
```

Chiffrement



Obfuscation

```
loc_401462: xor    eax, eax
loc_401464: xor    byte ptr [ebp+eax+var_2C], 2Ah
             inc    eax
             cmp    eax, 28h
             jl     short loc_401464
```

```
loc_401297: xor    byte ptr [ebp+eax+var_2C], 2Ah
             inc    eax
             cmp    eax, 28h
             jl     short loc_401297
loc_401464: lea    eax, [ebp+var_54]
             push   offset Dest
             push   eax
             pop    ecx
```

permet de cacher une fonction

```
.text:00401297 xor    byte ptr [ebp+eax+var_2C], 2Ah
.text:0040129C inc    eax
.text:0040129D cmp    eax, 28h
.text:004012A0 jl     short loc_401297
.text:004012A0 jl     short loc_401297
```

```
.text:00401464 xor    byte ptr [ebp+eax+var_2C], 2Ah
.text:00401469 inc    eax
.text:0040146A cmp    eax, 28h
.text:0040146D jl     short loc_401464
.text:0040146F push   offset aBravoTuAsTrouve ; "Bravo!!! Tu as trouv  le mot cach !!!"
```

```
.text:00401217 mov    [ebp+var_2C], 0A1C6A17Fh
.text:004012A8 lea    ecx, [ebp+var_2C]
.text:004012AE call   ecx
```

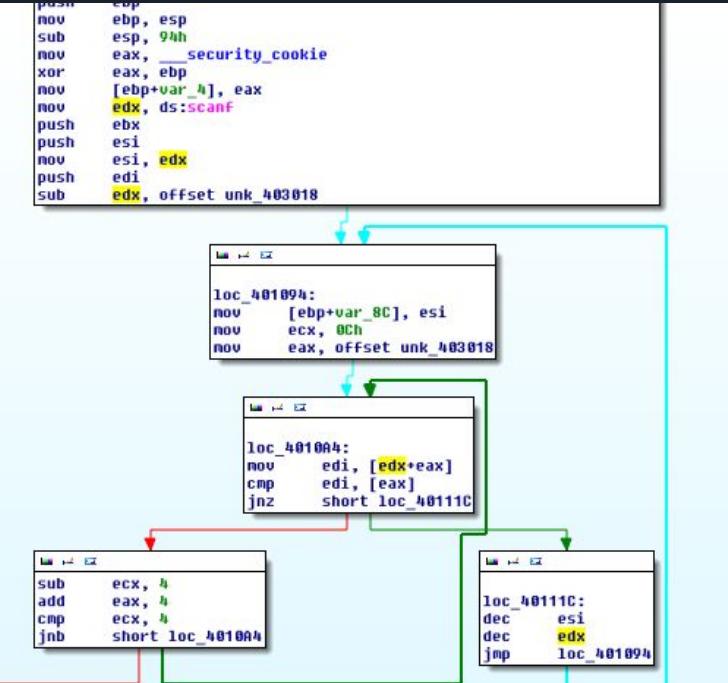
la fonction est charg e dans ecx, puis appeler

Obfuscation

```
RDX Stack[000005A4]:000000000012FF50 ; --
RIP Stack[000005A4]:000000000012FF50 push rbp
Stack[000005A4]:000000000012FF51 mov ebp, esp
Stack[000005A4]:000000000012FF53 mov ecx, [rbp+0Ch]
Stack[000005A4]:000000000012FF56 push rsi
Stack[000005A4]:000000000012FF57 mov esi, [rbp+8]
Stack[000005A4]:000000000012FF58 mov eax, 1
Stack[000005A4]:000000000012FF5F sub esi, ecx
Stack[000005A4]:000000000012FF61
Stack[000005A4]:000000000012FF61 loc_12FF61: ; CODE XREF: Stack[000005A4]:000000000012FF70†j
Stack[000005A4]:000000000012FF61 mov dl, [rsi+rax]
Stack[000005A4]:000000000012FF64 test dl, dl
Stack[000005A4]:000000000012FF66 jnz short loc_12FF6C
Stack[000005A4]:000000000012FF68 cmp [rcx], dl
Stack[000005A4]:000000000012FF6A jz short loc_12FF75
Stack[000005A4]:000000000012FF6C
Stack[000005A4]:000000000012FF6C loc_12FF6C: ; CODE XREF: Stack[000005A4]:000000000012FF66†j
Stack[000005A4]:000000000012FF6C cmp dl, [rcx]
Stack[000005A4]:000000000012FF6E jnz short loc_12FF73
Stack[000005A4]:000000000012FF70 jmp short loc_12FF61
Stack[000005A4]:000000000012FF73
Stack[000005A4]:000000000012FF73 loc_12FF73: ; CODE XREF: Stack[000005A4]:000000000012FF6E†j
Stack[000005A4]:000000000012FF73 xor eax, eax
Stack[000005A4]:000000000012FF75
Stack[000005A4]:000000000012FF75 loc_12FF75: ; CODE XREF: Stack[000005A4]:000000000012FF6A†j
Stack[000005A4]:000000000012FF75 pop rsi
Stack[000005A4]:000000000012FF76 pop rbp
Stack[000005A4]:000000000012FF77 ret
Stack[000005A4]:000000000012FF77 ; --
Stack[000005A4]:000000000012FF78 db 50h ; P
```

clef md5 : a618df803d5516a578d3407e7556241e

Obfuscation



recherche pointeur fonction à partir de scanf:

edx : printf
esi : edx
[ebp+var_8C] : esi

.data:00403018 unk_403018 db 6Ah ; j
.data:00403018 db 0Ch
.data:00403018 db 68h ; h
.data:00403018 db 60h
.data:00403018 db 57h ; W
.data:00403018 db 0B0h
.data:00403018 db 0B0h
.data:00403018 db 78h ; X
.data:00403018 db 0E8h ; p
.data:00403020 db 0C0h
.data:00403021 db 085h ; A
.data:00403022 db 0F0h
.data:00403023 db 0FFh
.data:00403024 db 0
.data:00403025 db 0
.data:00403026 db 0
.data:00403027 db 0

.text:004011CF push offset aleMotEntrsEstS ; "Le mot entré est : %s\n"
.text:004011D4 call esi

.text:004011E7 push offset aSonHashEnMd5Va ; "Son hash en MD5 vaut: %s\n"
.text:004011EC call esi

.text:004010C8 call esi

.text:0040143A call [ebp+var_8C]



Outils

IDA: graphes, analyse des instructions

Debugger de IDA



Résumé analyse du malware 2



Anti-debug utilisation d'anti debug au début



Obfuscation de fonction en partant de scanf et d'une fonction cacher



Chiffrement MD5 de base via wincrypt.h



Utilisation de plein de **variables négatives** pour remonter à partir de ebp