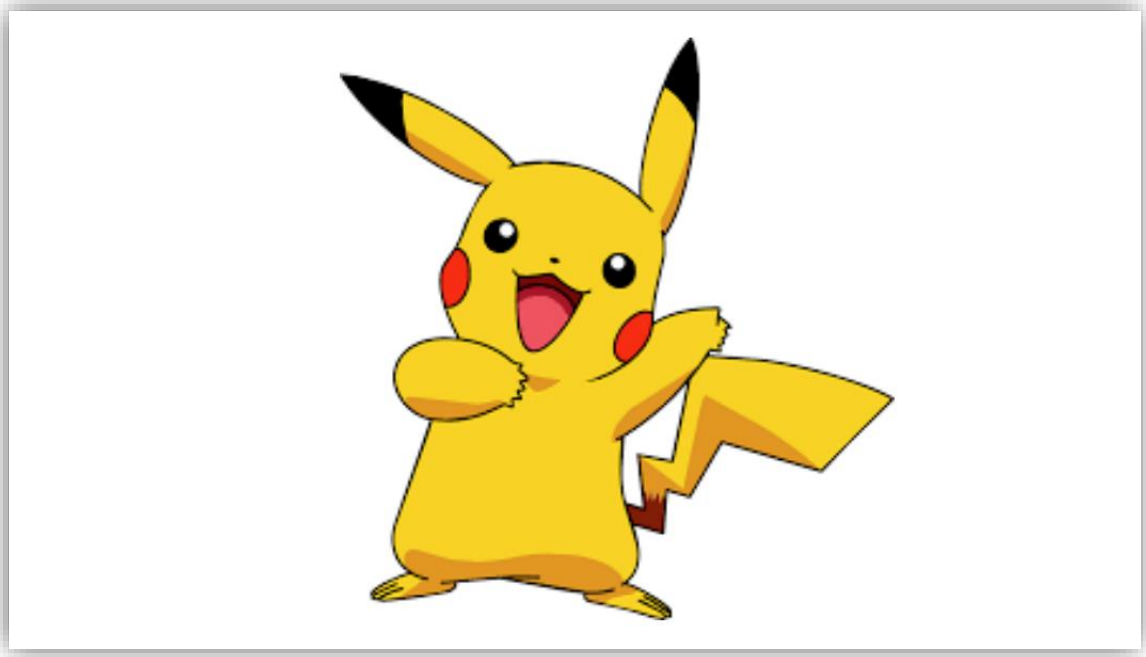
Université de Limoges

MASTER 2 Informatique – CRYPTIS :
Rapport de projet de Reverse



Présenté par :

- **Baptiste COUPRY**
- **Maxence HOUGUET**

Remis le 08 novembre 2024

I. x86-64

A. Découverte

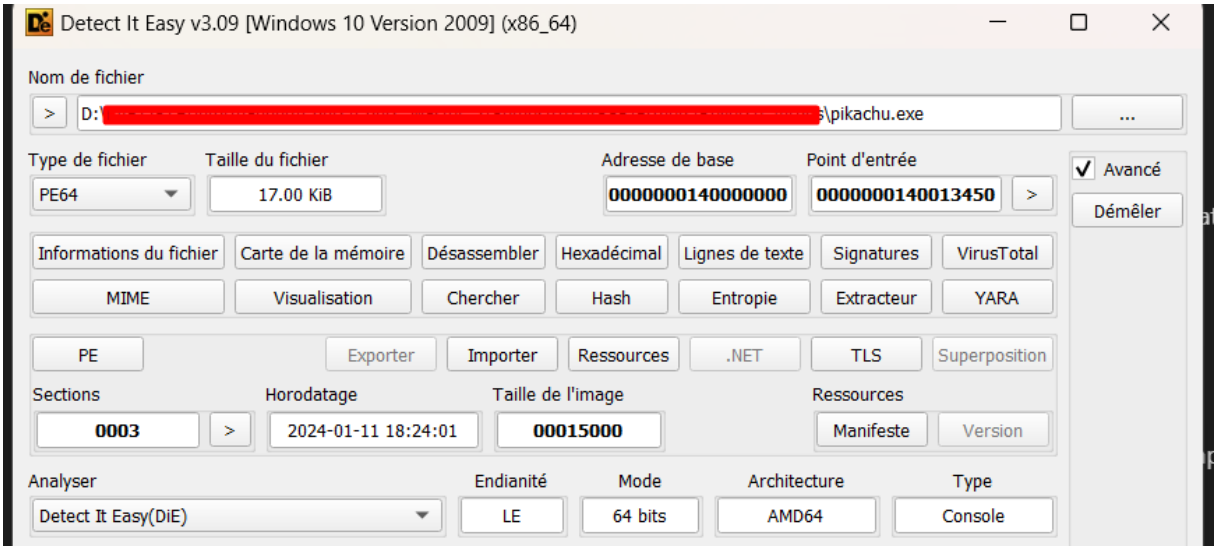
Dans la phase de découverte du binaire, nous devons tout d'abord analyser les méta-informations afin de déterminer par exemple si le binaire est compressé.

Questions :

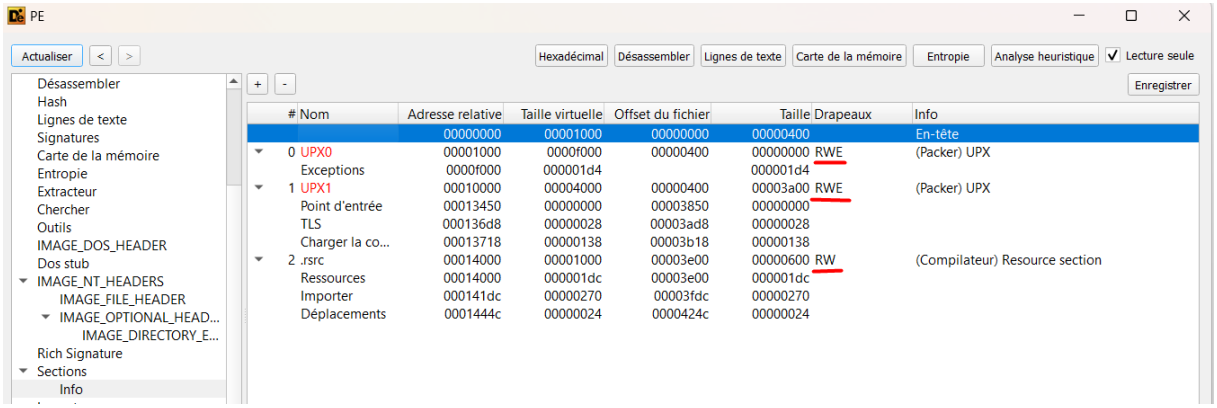
a. Quel est le timestamp contenu dans l'en-tête du binaire ?

Sigcheck est un outil de la suite Sysinternals qui permet d'analyser les métadonnées des fichiers du type PE. Utilisé en ligne de commande, il nous permet de retrouver notamment le timestamp de création du fichier avec ici le champs 'Link date', correspondant au timestamp de compilation, soit, la date à laquelle le fichier a été lié en un seul binaire exécutable.

Ce timestamp peut aussi se remarquer simplement avec l'outil Detect-it-Easy.



b. Combien de sections sont présentes dans le binaire ? Combien contiennent du code exécutable ?



On a ici 3 sections dont seules les 2 premières comportent le drapeau 'RWE'. Elles ont donc des accès mémoire en lecture, écriture, exécution, à contrario de la troisième section (.rsrc), qui n'a pas l'accès en exécution.

c. Quelle est la particularité de la première section? Que nous apprend sa raw size ?

Contrairement aux autres, la première section possède une Raw Size de 0x0, cela signifie qu'elle ne contient aucune information. Cela est dû à l'utilisation de certains logiciels de compression de fichier exécutable comme ici UPX, qui peut modifier la taille des sections.
En l'occurrence la première section a été modifiée par un logiciel de compression afin de ne contenir aucune donnée.

d. À l'aide de signatures, par exemple en utilisant le logiciel DiE ou CFF Explorer (présent dans la VM dans Explorer Suite), ou bien via le nom des sections, identifiez l'outil utilisé pour compresser le programme.

Comme vu dans la question précédente, l'outil de compression utilisé est UPX (Ultimate Packer for eXecutable).

<u>À partir de maintenant, nous travaillerons sur la version décompressée : vous devez donc décompresser le programme.</u>

Décompression de pikachu.exe à l'aide d'UPX :

```
baptoo@[redacted]$ upx -d pikachu.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2020
UPX 3.96        Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
     52736 <-     17408   33.01%    win64/pe     pikachu.exe

Unpacked 1 file.
```

B. <u>PointS d'entréeS</u>

La fonction ~~0x1400013E0~~ 0x140001400 est la fonction main du programme. Elle n'est pas pour autant le point d'entrée du programme, car c'est le CRT (C runtime) qui va être appelé comme point d'entrée afin de s'initialiser.

Questions :

a. Pouvez-vous décrire le fonctionnement de la fonction 0x140001400 ?

Fonction 0x140001400 :

```
; int __fastcall main(int argc, const char **argv, const char **envp)
main proc near

bInheritHandles= qword ptr -0D8h
dwCreationFlags= qword ptr -0D0h
lpEnvironment= qword ptr -0C8h
lpCurrentDirectory= qword ptr -0C0h
lpStartupInfo= dword ptr -0B8h
lpProcessInformation= qword ptr -0B0h
ProcessInformation= byte ptr -0A8h
StartupInfo= byte ptr -88h
var_18= qword ptr -18h

; __unwind { // __GSHandlerCheck
mov     r11, rsp
push    rbx
sub     rsp, 0F0h
mov     rax, cs:__security_cookie
xor     rax, rsp
mov     [rsp+0F8h+var_18], rax
xorps   xmm0, xmm0
lea     rcx, aLoading   ; "Loading...\n"
movups  xmmword ptr [rsp+0F8h+StartupInfo+4], xmm0
xor     eax, eax
movups  xmmword ptr [r11-74h], xmm0
mov     [r11-24h], eax
movups  xmmword ptr [r11-64h], xmm0
mov     [rsp+60h], rax
movups  xmmword ptr [r11-54h], xmm0
movups  xmmword ptr [r11-44h], xmm0
movups  xmmword ptr [r11-34h], xmm0
mov     dword ptr [rsp+0F8h+StartupInfo], 68h ; 'h'
movups  xmmword ptr [rsp+0F8h+ProcessInformation.Low], xmm0
0000000140001400: main
```

On s'assure que la fonction main est bien celle à l'adresse 0x140001400.
La fonction '0x140001400' (main) permet donc d'initialiser des protections de sécurité tel qu'ici, le security_cookie qui permet de détecter les débordements de tampon. Un cookie de sécurité est placé sur la pile et vérifié avant de quitter la fonction. Si le cookie a été modifié, cela indique une tentative d'attaque, ce qui peut entraîner l'arrêt du programme.
On reconnaît aussi l'affichage d'un message de chargement ("Loading...\n").
De la mémoire est aussi allouer et dans laquelle est chargé des données prédéfinies (celles de la chaîne 'aVwbyagkadablac' contenant l'affichage du Pikachu).
Après un délai, elle configure les structures 'ProcessInformation' et 'StartupInfo' qui sont nécessaires pour lancer un nouveau processus. La fonction appelle enfin `CreateProcessA` pour exécuter une commande avec les paramètres spécifiés. Si le lancement échoue, la fonction gère l'erreur en conséquence.

b. Quel est le processus responsable d'afficher Pikachu ?

```
call    sub_140001060
mov     ecx, 7D0h       ; dwMilliseconds
call    cs:Sleep
mov     ecx, 5043h      ; Size
call    cs:malloc
movaps  xmm0, cs:xmmword_140003290
lea     rdx, aVwbyagkadablac ; "VwByAGkAdABlAC0ATwB1AHQAcAB1AHQAIABAACI"...
mov     r8d, 5000h      ; Size
mov     rbx, rax
movups  xmmword ptr [rax], xmm0
movaps  xmm1, cs:xmmword_1400032A0
movups  xmmword ptr [rax+10h], xmm1
movaps  xmm0, cs:xmmword_1400032B0
movups  xmmword ptr [rax+20h], xmm0
movaps  xmm1, cs:xmmword_1400032C0
movups  xmmword ptr [rax+30h], xmm1
movzx   ecx, cs:word_1400032D0
mov     [rax+40h], cx
movzx   ecx, cs:byte_1400032D2
mov     [rax+42h], cl
lea     rcx, [rax+42h]  ; void *
call    memcpy
lea     rax, [rsp+0F8h+ProcessInformation]
xor     r9d, r9d        ; lpThreadAttributes
mov     [rsp+0F8h+lpProcessInformation], rax ; lpProcessInformation
xor     r8d, r8d        ; lpProcessAttributes
lea     rax, [rsp+0F8h+StartupInfo]
mov     rdx, rbx        ; lpCommandLine
mov     qword ptr [rsp+0F8h+lpStartupInfo], rax ; lpStartupInfo
xor     ecx, ecx        ; lpApplicationName
xor     eax, eax
mov     [rsp+0F8h+lpCurrentDirectory], rax ; lpCurrentDirectory
mov     [rsp+0F8h+lpEnvironment], rax ; lpEnvironment
mov     dword ptr [rsp+0F8h+dwCreationFlags], eax ; dwCreationFlags
mov     dword ptr [rsp+0F8h+bInheritHandles], eax ; bInheritHandles
call    cs:CreateProcessA
test    eax, eax
jz      short loc_140001534
```

Grâce à cette capture d'écran de la fonction main, il paraît très probable que le processus responsable d'afficher Pikachu soit 'CreateProcessA' qui pourrait être vu comme le processus résultant de l'appel à la fonction 'sub_140001060' :

```
sub_140001060 proc near

Format= qword ptr  8
arg_8= qword ptr  10h
arg_10= qword ptr  18h
arg_18= qword ptr  20h

mov     [rsp+Format], rcx
mov     [rsp+arg_8], rdx
mov     [rsp+arg_10], r8
mov     [rsp+arg_18], r9
push    rbx
sub     rsp, 20h
mov     ecx, 1          ; Ix
lea     rbx, [rsp+28h+arg_8]
call    cs:__acrt_iob_func
mov     rdx, [rsp+28h+Format] ; Format
mov     r9, rbx
mov     rcx, rax        ; Stream
call    sub_140001010
add     rsp, 20h
pop     rbx
retn
sub_140001060 endp
```

Cette fonction permet un affichage formaté de données dans le flux de sortie (accès au flux standard de sortie dit stdout : __acrt_iob_func).
Puisque cette fonction prend des arguments, il nous faut comprendre comment sont-ils passés et de quels types sont-ils. On se réfère donc à la fonction 'sub_140001010', appelée ici.

```
; __int64 __fastcall sub_140001010(FILE *Stream, char *Format)
sub_140001010 proc near

ArgList= qword ptr -18h
arg_0= qword ptr  8
arg_8= qword ptr  10h

mov     [rsp+arg_0], rbx
mov     [rsp+arg_8], rsi
push    rdi
sub     rsp, 30h
mov     rbx, r9
mov     rdi, rdx
mov     rsi, rcx
call    sub_140001000
xor     r9d, r9d        ; Locale
mov     [rsp+38h+ArgList], rbx ; ArgList
mov     r8, rdi         ; Format
mov     rdx, rsi        ; Stream
mov     rcx, [rax]      ; Options
call    cs:__stdio_common_vfprintf
mov     rbx, [rsp+38h+arg_0]
mov     rsi, [rsp+38h+arg_8]
add     rsp, 30h
pop     rdi
retn
sub_140001010 endp
```

La fonction sub_140001010 permet l'affichage formaté dans le stream indiqué grâce à '__stdio_common_vfprintf'. Ainsi, puisque 'sub_140001060e appelle 'sub_140001010', on peut comprendre que 'sub_140001060' prépare des données à afficher tandis que 'sub_140001010' exécute l'affichage. Affichage qui peut ici très bien être celui de Pikachu.

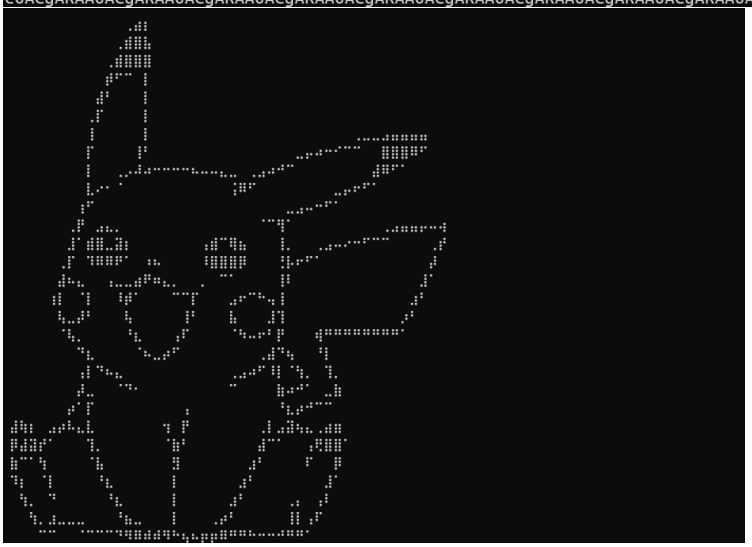
Par ailleurs, les données permettant l'affichage de Pikachu sont visibles au niveau de l'instruction « lea rdx, aVwbyagkadablac ; "VwByAGkAdABlAC0ATwB1AHQAcAB1AHQAIABAACI"... » où l'instruction 'lea' est utilisée ici pour charger l'adresse de la chaîne 'aVwbyagkadablac' dans le registre 'rdx'. Si on se rend sur cette chaîne, on obtient l'intégralité des chaînes en base 64 permettant l'affichage du Pikachu en UTF-16LE :

```
.rdata:00000001400032E0 aVwbyagkadablac db 'VwByAGkAdABlAC0ATwB1AHQAcAB1AHQAIABAACIACgAAKAAoACgAKAAoACgAKAAoA'
.rdata:00000001400032E0                                         ; DATA XREF: TlsCallback_0+146↑o
.rdata:00000001400032E0                                         ; TlsCallback_0+15D↑o ...
.rdata:0000000140003321                 db 'CgAKAAoACiAKPQoRigAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoAC'
.rdata:0000000140003362                 db 'gAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAoAACgAKAAoACgAKAAoACg'
.rdata:00000001400033A3                 db 'AKIAo/ij/KOcoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgA'
.rdata:00000001400033E4                 db 'KAAoACgAKAAoACgAKAAoACgAKAAoACgKAAAoACgAKAAoACgAKAAoACgAKAAogCj+K'
.rdata:0000000140003425                 db 'P8o/yj/KAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKA'
.rdata:0000000140003466                 db 'AoACgAKAAoACgAKAAoACgAKAAoCgAAKAAoACgAKAAoACgAKAAoACgAKH4oCygJKAA'
.rdata:00000001400034A7                 db 'oRygAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAo'
.rdata:00000001400034E8                 db 'ACgAKAAoACgAKAAoACgAKAoAACgAKAAoACgAKAAoACgAKAAo/CgDKAAoACgAKEcoA'
.rdata:0000000140003529                 db 'CgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoAC'
.rdata:000000014000356A                 db 'gAKAAoACgAKAAoACgKAAAoACgAKAAoACgAKAAoACiAKE8oACgAKAAoAChHKAAoACg'
.rdata:00000001400035AB                 db 'AKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgA'
.rdata:00000001400035EC                 db 'KAAoACgAKAAoCgAAKAAoACgAKAAoACgAKAAouCgAKAAoACgAKAAoRygAKAAoACgAK'
.rdata:000000014000362D                 db 'AAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACiAKMAowCjgKOQo5CjkKO'
.rdata:000000014000366E                 db 'QoACgAKAoAACgAKAAoACgAKAAoACgAKE8oACgAKAAoACi4KAMoACgAKAAoACgAKAA'
.rdata:00000001400036AF                 db 'oACgAKAAoACgAKAAoACgAKAAowChkKDQoEigKKAkoCSgAKAAo/yj/KP8oPygLKAAo'
.rdata:00000001400036F0                 db 'ACgKAAAoACgAKAAoACgAKAAoAChHKAAoACiAKGAoPCg0KBIoEigSKBIoJigkKCQox'
.rdata:0000000140003731                 db 'CjAKAAogCjgKDQoGigJKAAoACgAKAAoACgAKAAoACj8KD8oCygBKAAoACgAKAAoCg'
.rdata:0000000140003772                 db 'AAKAAoACgAKAAoACgAKAAoxygUKAIoCCgAKAAoACgAKAAoACgAKAAoqCg'
.rdata:00000001400037B3                 db '/KAsoACgAKAAoACgAKAAoACgAKMAoZCgWKAsoASgAKAAoACgAKAAoACgAKAoAACgA'
.rdata:00000001400037F4                 db 'KAAoACgAKAAoACiwKAsoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAK'
```

Ainsi, 'CreateProcessA' est exécuté pour permettre l'affichage de Pikachu.

```
if ( CreateProcessA(0uLL, shell_command, 0LL, 0LL, 0, 0, 0LL, 0LL, &StartupInfo, &ProcessInformation) )
{
  CloseHandle(ProcessInformation.hThread);
  WaitForSingleObject(ProcessInformation.hProcess, 0xFFFFFFFF);
  CloseHandle(ProcessInformation.hProcess);
}
```

```
PS D:[redacted]> powershell.exe -noprofile -execu
tionpolicy bypass -encodedcommand VwByAGkAdABlAC0ATwB1AHQAcAB1AHQAIABAACIACgAAKAAoACgAKAAoACgAKAAoACgAKAAoACiAKPQoRigAKA
AoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAo
coACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKP
```

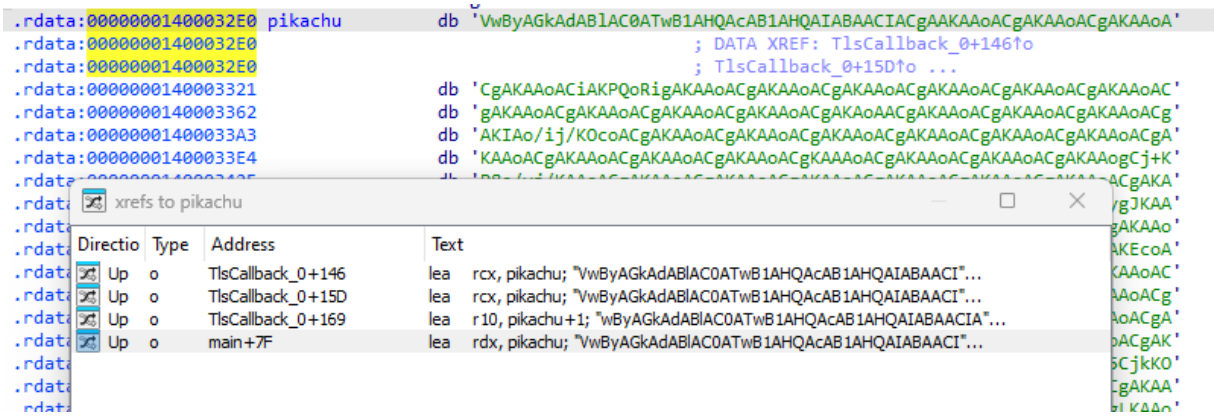


Voici le Pikachu affiché dans le terminal avec la commande trouvée dans le main : «powershell.exe -noprofile -executionpolicy bypass -encodedcommand [PAYLOAD]» :

```
13  strcpy(v3, "powershell.exe -noprofile -executionpolicy bypass -encodedcommand ");
```

c. Pouvez-vous identifier les fonctions qui font référence à la donnée en 0x1400032E0 ?

Afin de connaître quel sont les fonctions qui ont utilisé la donnée contenant le Pikachu, nous pouvons regarder les cross-reference :

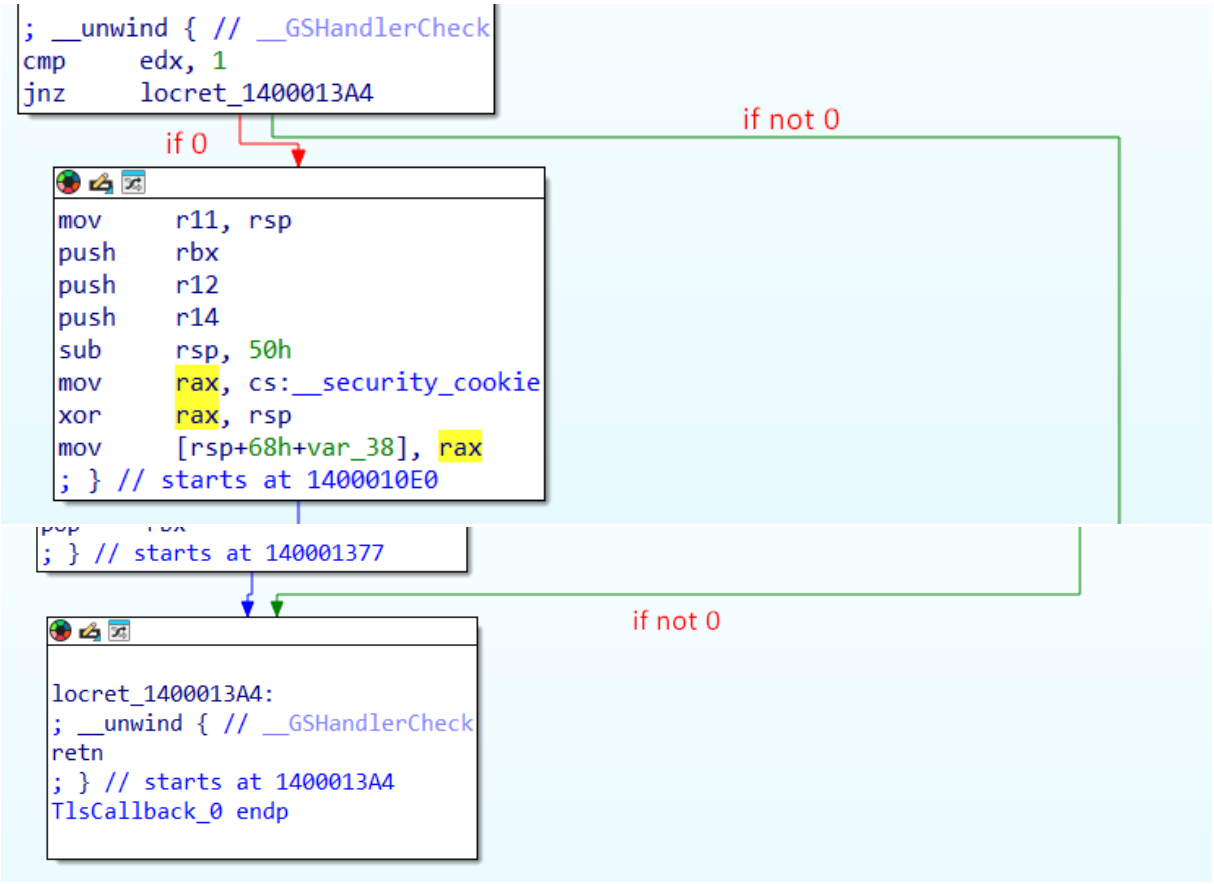


On y trouve 2 fonction, « main » et « TlsCallBack ».

d. Quel est le mécanisme qui exécute la fonction 0x1400010E0 ?

Le mécanisme qui exécute la fonction 0x1400010E0 (fonction TLSCallback_0) est le saut conditionnel à l'adresse 'locret_1400013A4' donné par l'instruction 'jnz locret_1400013A4'. En effet, l'instruction 'cmp edx, 1' compare la valeur du registre 'edx' avec la valeur '1'. Ensuite, l'instruction 'jnz locret_1400013A4' indique qu'il y a un saut conditionnel à l'adresse 'locret_1400013A4' si le registre 'edx' ne vaut pas 1. L'instruction 'jnz' signifie « jump if not zero ». Les captures d'écran ci-dessous illustrent ce fonctionnement.

Note : L'adresse 0x1400010E0 renvoie vers l'instruction 'cmp edx, 1'.



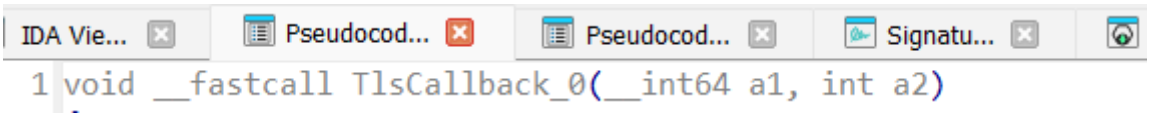
e. Qui est exécuté en premier entre la fonction 0x140001400 et 0x1400010E0 ?

Address	Function	Instruction
.text:00000001400010E0	TlsCallback_0	
.text:00000001400010FF	TlsCallback_0	; } // starts at 1400010E0

La fonction 0x1400010E0 (TLSCallback_0) est exécutée en premier, avant 0x140001400 (main).
La fonction de rappel TLSCallback_0 (TLS pour Thread Local Storage) est déclenchée automatiquement par le système avant le main au moment du démarrage du programme. Elle permet de configurer des données spécifiques à un thread qui vient s'exécuter. Cette fonction peut aussi être utilisée par des acteurs malveillants pour rendre l'analyse et la détection des événements au sein de leurs applications plus compliquée.
De cette façon, cela garanti que tout ce qui doit être configuré pour les threads l'est bien, avant que le main ne puisse s'exécuter.

f. Donnez la signature de la fonction 0x1400010E0.

La signature de la fonction 0x1400010E0 donnée par IDA est 'public TlsCallback_0' que l'on peut trouver si l'on affiche le pseudo-code du programme.



Métamorph

D'après la section précédente, nous savons que la fonction 0x1400010E0 est la fonction dite "backdoor". Cette dernière prend comme premier argument l'adresse de base du module courant (HMODULE), soit pikachu.exe.

Questions :

a. Quel est le nom de l'algorithme implémenté dans la fonction 0x1400013B0 ?

Le pseudo-code de l'algorithme implémenté dans la fonction 0x1400013B0 est :

```
1  __int64 __fastcall sub_1400013B0(char *a1, unsigned __int64 a2)
2  {
3    __int64 result; // rax
4    int i; // r10d
5    int v5; // ecx
6    int v6; // eax
7
8    result = 0LL;
9    for ( i = 0; i < a2; result = (32 * (unsigned __int8)v6) ^ (v6 << 12) ^ (unsigned int)v6 )
10   {
11     v5 = *a1++;
12     ++i;
13     v6 = ((unsigned __int8)(v5 ^ BYTE1(result)) >> 4) ^ v5 ^ (BYTE1(result) | ((_DWORD)result << 8));
14   }
15   return result;
16 }
```

Ainsi, on peut comprendre que la fonction sub_1400013B0 implémente un algorithme de hachage qui va être appliqué une chaîne de caractères (soit, un tableau d'octets) dont le pointeur est passé en argument (a1), ainsi que la longueur de la chaîne a1 (a2). En effet, on peut le comprendre suivant le fonctionnement de l'algorithme qui est, succinctement :

- Initialisation :
  - L'entier de 64 bits 'result' est initialisé à 0.
- Fonctionnement de l'algorithme :
  - La fonction itère sur chaque octet de l'entrée donnée ('a1') des opérations bit à bit avec des XOR et des décalages.
- Retour :
  - La fonction retourne 'result' qui représenterait le hash de l'entrée donnée ('a1').

b. Quels sont les noms des deux fonctions qui sont résolues dans la fonction 0x1400010A0 ?

```
1  struct _LIST_ENTRY *sub_1400010A0()
2  {
3    int i; // ecx
4    struct _PEB_LDR_DATA *Ldr; // rdx
5    struct _LIST_ENTRY *var_Flink; // rax
6    struct _LIST_ENTRY *p_InMemoryOrderModuleList; // rdx
7
8    i = 0;
9    Ldr = NtCurrentPeb()->Ldr;
10   var_Flink = Ldr->InMemoryOrderModuleList.Flink;
11   p_InMemoryOrderModuleList = &Ldr->InMemoryOrderModuleList;
12   if ( var_Flink == p_InMemoryOrderModuleList )
13     return 0LL;
14   while ( i != 2 )
15   {
16     var_Flink = var_Flink->Flink;
17     ++i;
18     if ( var_Flink == p_InMemoryOrderModuleList )
19       return 0LL;
20   }
21   return var_Flink[2].Flink;
22 }
```

La fonction ‘sub_1400010A0()’ permet d’accéder au 3ème module chargé, à l’aide de la liste ‘InMemoryOrderModuleList’ dans le PEB (Process Environment Block). Le PEB est une structure de données utilisée par Windows contenant des informations sur chaque processus (chaque processus à son PEB associé) en cours d'exécution telle qu’ici, la liste des modules chargés.

Un malware a tout intérêt à passer par le PEB pour accéder à des informations sur les processus. Cela lui évite de passer par des API telles que ‘GetProcAddress’ ou ‘LoadLibrary’ où l’activité malicieuse serait détectée par les antivirus.

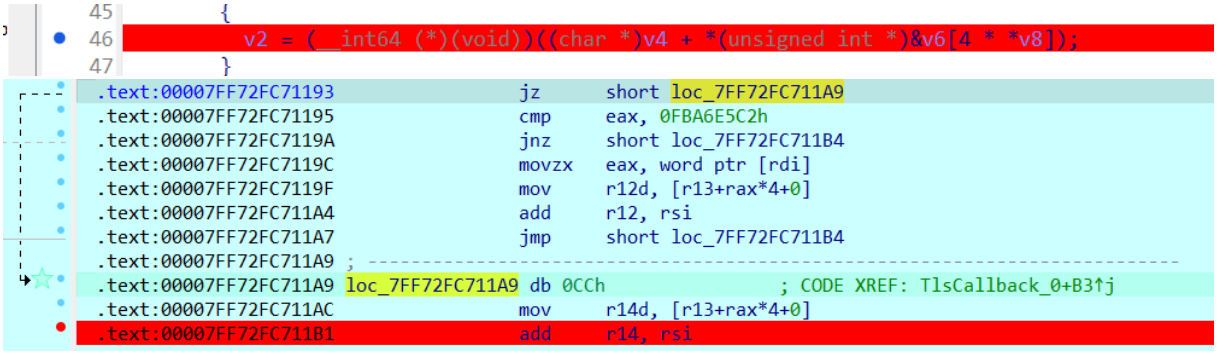
En comprenant ce programme, on comprend donc quels sont les 2 fonctions résolues ici, à savoir :

- **NtCurrentPeb() :** Cette fonction permet d’obtenir un pointeur vers le PEB du processus en cours.

InMemoryOrderModuleList : Ici est contenue la liste des modules chargés par le processus en fonction de leur ordre de chargement en mémoire.

Le pointeur **’Flink’** permet, lui, d’accéder aux modules chargés par le processus.

Cependant, on ne sait pas vraiment s’il est évoqué cette fonction OU la fonction ‘TlsCallback_0’.
Dans le doute, on émet l’hypothèse que les 2 fonctions résolues dans ‘TlsCallback_0’ sont :
La fonction (reconnue par le ‘void’) à la ligne 46 de la fonction :
Ajout du breakpoint :



On reconnaît aussi le ‘if’ qui est ici ‘loc_7FF72FC711A9’ (instruction conditionnel (jz)).

Notons que l'opération est mise dans le registre r14, qui a été initialisé à 0 grâce à l'opération :

```
.text:00007FF72FC71116                  xor     r14d, r14d
```

Ici aussi serait probablement la seconde fonction résolue :

```
47          {
48            v3 = (void (__fastcall *)(const char *, __int64, __int64, __int64 *))(v4 + *(unsigned int *)(v6 + 4LL * *v8));
49          }
```

```
.text:00007FF72FC7119F                  mov     r12d, [r13+rax*4+0]
.text:00007FF72FC711A4                  add     r12, rsi
```

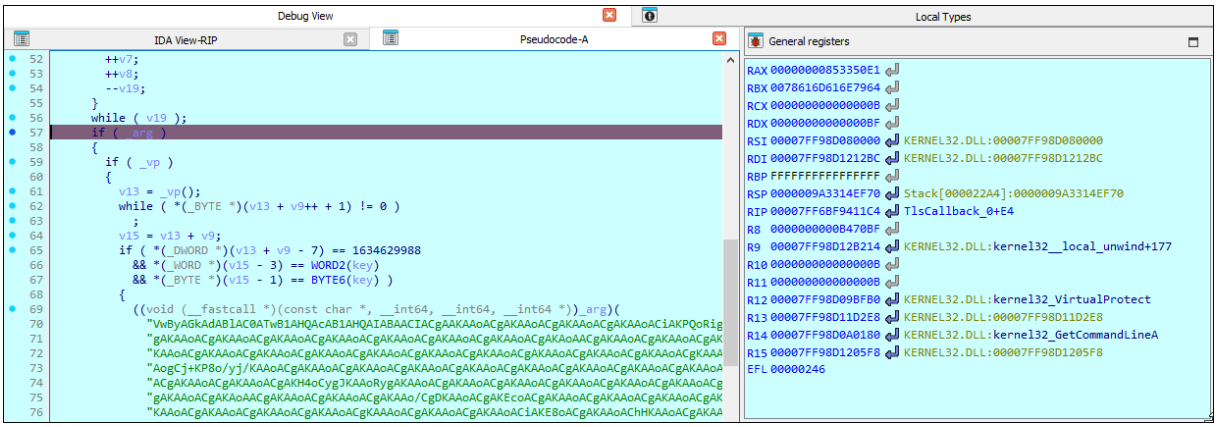
En effet, le registre r12 subit lui aussi une initialisation à 0 à l'aide de r14 :

```
.text:00007FF72FC7111D                  mov     r12d, r14d
```

Ainsi, nous pensons v2 et v3 liés et de fait que v2 appelle probablement une fonction, alors v3 en appellerait aussi une qui pourrait être présente dans la même dll.

Conclusion :

En lançant le debugger on reconnaît en effet la fonction VirtualProtect ainsi que GetCommandLineA de kernel32.dll.



De fait, on peut conclure que les 2 fonctions résolues sont ces dernières.

c. Donner l'algorithme, en C, utilisé pour résoudre l'adresse de ces deux fonctions.

Cas 1 - fonctions dans v2 et v3 :

Le code C résolvant l'adresse de ces 2 fonctions serait alors :

```
[redacted] > gcc addr.c -o addr.exe
[redacted] > ./addr.exe kernel32.dll Vir
tualProtect
Adresse de VirtualProtect dans kernel32.dll : 00007FFE5F415470
[redacted] > ./addr.exe kernel32.dll Get
CommandLineA
Adresse de GetCommandLineA dans kernel32.dll : 00007FFE5F418AD0
```

```c
#include <windows.h>
#include <stdio.h>

void* get_addr_fonction(const char* module_dll, const char* fonction) {

    // On charge ici le module correspondant avec LoadLibraryA
    HMODULE module_dll_charge = LoadLibraryA(module_dll);
    // on obtient l'@ avec GetProcAddress qui recupere l'adresse d'une fonction exportee
    void * addr_fonction = GetProcAddress(module_dll_charge, fonction);

    return addr_fonction;
}

int main(int argc, char* argv[]) {

    if (argc < 3) {
        printf("Utilisation du prog : %s <module_dll> <fonction>", argv[0]);
        exit(0);
    }

    const char* module_dll = argv[1];
    const char* fonction = argv[2];

    void *addr_function = get_addr_fonction(module_dll, fonction);

    if (addr_function) {
        printf("Adresse de %s dans %s : %p\n", fonction, module_dll, addr_function);
    }
    else {
        printf("Adresse de %s dans %s impossible à retrouver\n", fonction, module_dll);
    }

    return 0;
}

/* Notes :
```

Note :

On remarquera ici que la résolution des adresses ne se fait pas selon le programme pikachu.exe. Une DLL a une adresse de chargement préférée qui est définie dans le champ 'ImageBase' de la structure 'IMAGE_OPTIONAL_HEADER'. Au moment où l'OS charge la dll avec ses fonctions, il va le faire à l'adresse préférée dans la mesure du possible, sinon la dll sera déplacée à une autre adresse. Ainsi, ce programme permet donc de récupérer l'adresse de ces fonctions une fois la dll chargée (hors du programme pikachu.exe).

Le programme est disponible sous le nom 'addr.c' dans l'archive zip du projet.

Cas 2 - fonction NtCurrentPeb :

Le code C résolvant l'adresse de cette fonction serait alors :

```
Ldr = NtCurrentPeb()->Ldr;
Flink = Ldr->In
p_InMemoryOrder
```
struct _PEB *(__fastcall __pure *)()

d. Quelle est la taille de la donnée qui commence à l'adresse 0x1400082E0, référencée dans la fonction 0x1400010E0 ?

```
106   memcpy(
107     "VwByAGkAdABlAC0ATwB1AHQAcAB1AHQAIABAACIACgAAKAAoACgAKAAoACgAKAAoACgAKAAoACiAKPQoRigAKAAoACgAKAAoACgAKAAoAC"
108     "gAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKIAo/ij/KOcoACgA"
140   "gAKAAouChHKKAoDygAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgAKAAoACgIg"
141   "KHYodig/KBsoGygTKBIoEigaKBsoGygBKAoAIgBAAA==",
142   &unk_1400082E0,
143   0x5000uLL);
```

La taille de la donnée est de '0x5000uLL', soit une représentation d'un nombre en C :

- 0x : Indique que le nombre est en hexadécimal.
- 5000 : En hexadécimal, cela correspond à 20480 en décimal.
- u : Indique que le nombre est de type unsigned.
- LL : Indique que le nombre est de type 'long long'.

Donc, '0x5000uLL' équivaut à 20480 octets, de type unsigned long long.

Toutefois, si l'on cherche plus loin dans le code, on trouve :

```
for ( i = 0; i < 19392; i += 6 )
{
  *(v16 - 1) ^= *((_BYTE *)&v20 + i % 7u);
  *v16 ^= *((_BYTE *)&v20 + (int)(i - 7 * ((i + 1) / 7u) + 1));
  v16[1] ^= *((_BYTE *)&v20 + (int)(i - 7 * ((i + 2) / 7u) + 2));
  v16[2] ^= *((_BYTE *)&v20 + (int)(i - 7 * ((i + 3) / 7u) + 3));
  v16[3] ^= *((_BYTE *)&v20 + (int)(i - 7 * ((i + 4) / 7u) + 4));
  v18 = (int)(i - 7 * ((i + 5) / 7u) + 5);
  v16 += 6;
  *(v16 - 2) ^= *((_BYTE *)&v20 + v18);
}
```

Ce qui nous indique qu'en tout cas, la taille du PowerShell à retrouver ici est de 19392 octets.

e. Quelle modification est appliquée aux données présentes à l'adresse 0x1400082E0 ?

La modification apportée aux données présentes à l'adresse '0x1400082E0' est le fait qu'il n'y a pas la lettre 'V' dans l'encodage du Pikachu en b64. Voir image ci-dessous dans 'v15'.

```
  &unk_7FF72FC782E0,
  0x5000uLL);
v16 = "wByAGkAdABlAC0ATwB1AHQAcAB1AHQAIABAACIACgAAKAAoACgAKAAoACgAKAAoAC
for ( i = 0; i < 19392; i += 6 )
{
```

Le premier octet de la chaîne a été supprimé (un décalage a en réalité été effectué qui pourrait se traduire ainsi : « v16 = (byte *)(pikachu_b64 + 1); »).
Toutefois, dans la boucle for, dès la première instruction, on redécale la chaîne de 1 octet dans le sens opposé, ce qui annule le premier décalage. Ainsi, aucune modification n'est directement appliquée aux données à l'adresse '0x1400082E0'.

f. Sous quelle condition la commande PowerShell est-elle réécrite ?

la condition pour réecrire le powershell est d'ajouter la clé « dynamax » en argument au lancement du programme :

```
if ( *(_DWORD *)(chaine1 + chaine2 - 7) == 'anyd'
  && *(_WORD *)(chaine12 - 3) == WORD2(dynamax)
  && *(_BYTE *)(chaine12 - 1) == BYTE6(dynamax) )
{
  ((void (__fastcall *)(const char *, __int64, __int64, __int64 *))list1)(
```

g. Pourquoi sommes-nous obligés d'utiliser la fonction VirtualProtect, qui permet de changer les droits d'une zone mémoire ?

La fonction 'VirtualProtect' permet de modifier les paramètres de protection (xrw, xr-, …) de l'application. Cela signifie qu'il est possible d'aller modifier les permissions d'accès, de l'application, à une zone de mémoire déjà allouée. Ainsi,

Le code ci-contre est une illustration permettant de montrer comment VirtualProtect peut être utilisée pour changer les droits de la mémoire pour permettre l'exécution :

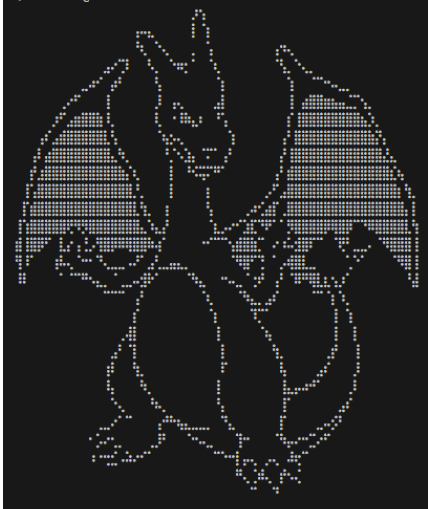
```
if (!VirtualProtect(v3, 0x5043uLL, PAGE_EXECUTE_READWRITE, &pdword_oldProtection)) {
    fprintf(stderr, "Err lors de l'appel de la fonction 'VirtualProtect'.\n");
    return 1;
}
```

La constante de protection de la mémoire 'PAGE_EXECUTE_READWRITE' permet d'activer l'accès à l'exécution, en lecture seule ou en lecture/écriture à la région validée des pages. Ainsi, puisque la fonction 'VirtualProtect' est ici utilisée (voir question b.), la commande devrait pouvoir s'exécuter même si des restrictions seraient posées par le système.

h. Faire un script Python decode.py permettant de décoder le Powershell Dracaufeu present à l'adresse ~~0x1400082F0~~ 0x1400082E0. Votre script doit s'exécuter de la manière suivante : python decode.py pikachu.exe dracaufeu.ps1

Script python disponible dans l'archive.
En ayant décodé le powershell avec le script python écrit, on obtient :

```
--------------------------------------------------------------------------------
(.((1+1)). iex (function 1+1($b, $k) {$n = New-Object System.Collections.Generic.List[System.Object];for ($i = 0; $i -lt $b.Count; $i++) {$n.Add([char]($b[$i] -bxo
r $k[$i]))} return -join $n.ToArray();})
@(3,0,8,23,4,88,41,16,1,94,16,12). iex (function 1+1($b, $k) {$n = New-Object System.Collections.Generic.List[System.Object];for ($i = 0; $i -lt $b.Count; $i++) {$
n.Add([char]($b[$i] -bxor $k[$i]))} return -join $n.ToArray();})
 . Get-WmiObject Win32_Process -Filter "ProcessId = '$( . Get-WmiObject Win32_Process -Filter "ProcessId = ' [System.Diagnostics.Process]::GetCurrentProcess() | .
Select-Object -ExpandProperty ID'".ParentProcessId)'".(ProcessName));
try {
&$Wrm8EMP @"
```



```
"@
}
catch {
        . Write-Output (Pikachu n est pas disponible en dynamax !!!)
```

C. Powershell

Questions :

a. Donner la liste des cmdlets qui sont obfusqués au travers de l'opérateur d'invocation ˙ ou & ?

Voici les cmdlet obfusquées :

```
$dQRWGpP=83;$m83ytj3=101;$JZbPRdx=108;$2bGFs0n=99;$IB7a8ZE=116;$4JNB57M=45;$WdlS4IE=79;$7RCpZcf=98;$vqEpUU1=106;$9lW0iqg=71;$BmmmmZl=87;$yIaqt4f=109;$hD0mE1u=105;$v6
toWZf=120;$3Mk19cu=49;$Ybe2ueL=43;$pOFj7MC=51;$G3WdcJX=0;$WtUReU3=8;$Cl0KEnn=23;$fmKyDnj=4;$BndBQNw=88;$TqzS5S5=41;$YnjQuXN=16;$YLUW40K=1;$qVWLerF=94;$zvAxZf5=12;$6G
mjwOb=80;$KDIAD6V=114;$9GveMwq=111;$G7EUYfF=115;$A6dkVH0=78;$VJgdIv9=97;$UqZ8QWd=117;$xxoESSt=112;$zr8KNJC=102;$l20rjA9=110;$1OTzqiy=32;$6iWzL9I=40;$TKZvI17=36;$VuGQ
JOr=44;$u49DQDf=107;$Ai8VfwJ=123;$6W8EWAf=61;$WcMv5Ve=119;$SDR6Leb=121;$JX9l8Ad=46;$8TpJ1U6=67;$5zXgGI9=76;$BR44MnX=91;$SjKUbrN=93;$GMM0VAk=59;$9hKyK1w=48;$5CCRyVN=6
5;$FL9a7vL=100;$v3OZPNF=104;$OeRF5lQ=125;$hyeGZZS=84;$PKAOYmh=33;$QCZYyAR=([char]$dQRWGpP+[char]$m83ytj3+[char]$JZbPRdx+[char]$m83ytj3+[char]$2bGFs0n+[char]$IB7a8ZE+
[char]$4JNB57M+[char]$WdlS4IE+[char]$7RCpZcf+[char]$vqEpUU1+[char]$m83ytj3+[char]$2bGFs0n+[char]$IB7a8ZE);$5sqiujS=([char]$9lW0iqg+[char]$m83ytj3+[char]$IB7a8ZE+[cha
r]$4JNB57M+[char]$BmmmmZl+[char]$yIaqt4f+[char]$hD0mE1u+[char]$WdlS4IE+[char]$7RCpZcf+[char]$vqEpUU1+[char]$m83ytj3+[char]$2bGFs0n+[char]$IB7a8ZE);$GoU0UFd=([char]$9
lW0iqg+[char]$m83ytj3+[char]$IB7a8ZE+[char]$4JNB57M+[char]$BmmmmZl+[char]$yIaqt4f+[char]$hD0mE1u+[char]$WdlS4IE+[char]$7RCpZcf+[char]$vqEpUU1+[char]$m83ytj3+[char]$2
bGFs0n+[char]$IB7a8ZE);$WdrZONR=([char]$hD0mE1u+[char]$m83ytj3+[char]$v6toWZf);$KOEurzh=([char]$BmmmmZl+[char]$KDIAD6V+[char]$hD0mE1u+[char]$IB7a8ZE+[char]$m83ytj3+[
char]$4JNB57M+[char]$WdlS4IE+[char]$UqZ8QWd+[char]$IB7a8ZE+[char]$xxoESSt+[char]$UqZ8QWd+[char]$IB7a8ZE);$id = [System.Diagnostics.Process]::GetCurrentProcess() | .
$QCZYyAR -ExpandProperty ID
$dQRWGpQ = . $5sqiujS Win32_Process -Filter "ProcessId = '$id'"
$dQRWGpQ = . $GoU0UFd Win32_Process -Filter "ProcessId = '$($dQRWGpQ.ParentProcessId)'"

. $WdrZONR ([char]$zr8KNJC+[char]$UqZ8QWd+[char]$l20rjA9+[char]$2bGFs0n+[char]$IB7a8ZE+[char]$hD0mE1u+[char]$9GveMwq+[char]$l20rjA9+[char]$1OTzqiy+[char]$3Mk19cu+[ch
ar]$Ybe2ueL+[char]$3Mk19cu+[char]$6iWzL9I+[char]$TKZvI17+[char]$7RCpZcf+[char]$YuGQJOr+[char]$1OTzqiy+[char]$TKZvI17+[char]$u49DQDf+[char]$TqzS5S5+[char]$1OTzqiy+[ch
ar]$Ai8VfwJ+[char]$TKZvI17+[char]$l20rjA9+[char]$1OTzqiy+[char]$6W8EWAf+[char]$1OTzqiy+[char]$A6dkVH0+[char]$m83ytj3+[char]$WcMv5Ve+[char]$4JNB57M+[char]$Wdl S4IE+[ch
ar]$7RCpZcf+[char]$vqEpUU1+[char]$m83ytj3+[char]$2bGFs0n+[char]$IB7a8ZE+[char]$1OTzqiy+[char]$dQRWGpP+[char]$SDR6Leb+[char]$G7EUYfF+[char]$IB7a8ZE+[char]$m83ytj3+[ch
ar]$yIaqt4f+[char]$JX9l8Ad+[char]$8TpJ1U6+[char]$9GveMwq+[char]$JZbPRdx+[char]$JZbPRdx+[char]$m83ytj3+[char]$2bGFs0n+[char]$IB7a8ZE+[char]$hD0mE1u+[char]$9GveMwq+[ch
ar]$l20rjA9+[char]$G7EUYfF+[char]$JX9l8Ad+[char]$9lW0iqg+[char]$m83ytj3+[char]$l20rjA9+[char]$m83ytj3+[char]$KDIAD6V+[char]$hD0mE1u+[char]$2bGFs0n+[char]$JX9l8Ad+[ch
ar]$5zXgGI9+[char]$hD0mE1u+[char]$G7EUYfF+[char]$IB7a8ZE+[char]$BR44MnX+[char]$dQRWGpP+[char]$SDR6Leb+[char]$G7EUYfF+[char]$IB7a8ZE+[char]$m83ytj3+[char]$yIaqt4f+[ch
ar]$JX9l8Ad+[char]$WdlS4IE+[char]$7RCpZcf+[char]$vqEpUU1+[char]$m83ytj3+[char]$2bGFs0n+[char]$IB7a8ZE+[char]$SjKUbrN+[char]$GMM0VAk+[char]$zr8KNJC+[char]$9GveMwq+[ch
ar]$KDIAD6V+[char]$1OTzqiy+[char]$6iWzL9I+[char]$TKZvI17+[char]$hD0mE1u+[char]$1OTzqiy+[char]$6W8EWAf+[char]$1OTzqiy+[char]$9hKyK1w+[char]$GMM0VAk+[char]$1OTzqiy+[ch
ar]$TKZvI17+[char]$hD0mE1u+[char]$1OTzqiy+[char]$4JNB57M+[char]$JZbPRdx+[char]$IB7a8ZE+[char]$1OTzqiy+[char]$TKZvI17+[char]$7RCpZcf+[char]$JX9l8Ad+[char]$8TpJ1U6+[ch
ar]$9GveMwq+[char]$UqZ8QWd+[char]$l20rjA9+[char]$IB7a8ZE+[char]$GMM0VAk+[char]$1OTzqiy+[char]$TKZvI17+[char]$hD0mE1u+[char]$Ybe2ueL+[char]$Ybe2ueL+[char]$TqzS5S5+[ch
ar]$1OTzqiy+[char]$Ai8VfwJ+[char]$TKZvI17+[char]$l20rjA9+[char]$JX9l8Ad+[char]$5CCRyVN+[char]$FL9a7vL+[char]$FL9a7vL+[char]$6iWzL9I+[char]$BR44MnX+[char]$2bGFs0n+[ch
ar]$v3OZPNF+[char]$VJgdIv9+[char]$KDIAD6V+[char]$SjKUbrN+[char]$6iWzL9I+[char]$TKZvI17+[char]$7RCpZcf+[char]$BR44MnX+[char]$TKZvI17+[char]$hD0mE1u+[char]$SjKUbrN+[ch
ar]$1OTzqiy+[char]$4JNB57M+[char]$7RCpZcf+[char]$v6toWZf+[char]$9GveMwq+[char]$KDIAD6V+[char]$1OTzqiy+[char]$TKZvI17+[char]$u49DQDf+[char]$BR44MnX+[char]$TKZvI17+[ch
ar]$hD0mE1u+[char]$SjKUbrN+[char]$TqzS5S5+[char]$TqzS5S5+[char]$OeRF5lQ+[char]$1OTzqiy+[char]$KDIAD6V+[char]$m83ytj3+[char]$IB7a8ZE+[char]$UqZ8QWd+[char]$KDIAD6V+[ch
ar]$l20rjA9+[char]$1OTzqiy+[char]$4JNB57M+[char]$vqEpUU1+[char]$9GveMwq+[char]$hD0mE1u+[char]$l20rjA9+[char]$1OTzqiy+[char]$TKZvI17+[char]$l20rjA9+[char]$JX9l8Ad+[ch
ar]$hyeGZZS+[char]$9GveMwq+[char]$5CCRyVN+[char]$KDIAD6V+[char]$KDIAD6V+[char]$VJgdIv9+[char]$SDR6Leb+[char]$6iWzL9I+[char]$TqzS5S5+[char]$GMM0VAk+[char]$OeRF5lQ)
$Wrm8EMP=(.([char]$3Mk19cu+[char]$Ybe2ueL+[char]$3Mk19cu))
@($pOFj7MC,$G3WdcJX,$WtUReU3,$Cl0KEnn,$fmKyDnj,$BndBQNw,$TqzS5S5,$YnjQuXN,$YLUW40K,$qVWLerF,$YnjQuXN,$zvAxZf5) $dQRWGpQ.([char]$6GmjwOb+[char]$KDIAD6V+[char]$9GveMwq
+[char]$2bGFs0n+[char]$m83ytj3+[char]$G7EUYfF+[char]$G7EUYfF+[char]$A6dkVH0+[char]$VJgdIv9+[char]$yIaqt4f+[char]$m83ytj3));
try {
&$Wrm8EMP @"
```

```
"@
}
catch {
    . $KOEurzh ([char]$6GmjwOb+[char]$hD0mE1u+[char]$u49DQDf+[char]$VJgdIv9+[char]$2bGFs0n+[char]$v3OZPNF+[char]$UqZ8QWd+[char]$1OTzqiy+[char]$l20rjA9+[char]$1OTzqiy
    +[char]$m83ytj3+[char]$G7EUYfF+[char]$IB7a8ZE+[char]$1OTzqiy+[char]$xxoESSt+[char]$VJgdIv9+[char]$G7EUYfF+[char]$1OTzqiy+[char]$FL9a7vL+[char]$hD0mE1u+[char]$G7E
    UYfF+[char]$xxoESSt+[char]$9GveMwq+[char]$l20rjA9+[char]$hD0mE1u+[char]$7RCpZcf+[char]$JZbPRdx+[char]$m83ytj3+[char]$1OTzqiy+[char]$m83ytj3+[char]$l20rjA9+[char]
    $1OTzqiy+[char]$FL9a7vL+[char]$SDR6Leb
    +[char]$l20rjA9+[char]$VJgdIv9+[char]$yIaqt4f+[char]$VJgdIv9+[char]$v6toWZf+[char]$1OTzqiy+[char]$PKAOYmh+[char]$PKAOYmh+[char]$PKAOYmh)
}
```

Voici les cmdlet désobfusquées :

```
.((1+1)). iex (function 1+1($b, $k) {
    $n = New-Object System.Collections.Generic.List[System.Object];
    for ($i = 0; $i -lt $b.Count; $i++) {
        $n.Add([char]($b[$i] -bxor $k[$i]))
        }
    return -join $n.ToArray();
}
$Wrm8EMP = (.((1+1))@(3,0,8,23,4,88,41,16,1,94,16,12) . Get-WmiObject Win32_Process -Filter "ProcessId = '$( . Get-WmiObject Win32_Process -Filter "ProcessId = ' [
System.Diagnostics.Process]::GetCurrentProcess() | . Select-Object -ExpandProperty ID'".ParentProcessId)'".(ProcessName));
try {
&$Wrm8EMP @"

"@
}
catch {
    . Write-Output (Pikachu n est pas disponible en dynamax !!!)
}
```

b. Quelle information est stockée dans le variable $dqrwgpq après le dernier appel à la cmdlet Get-Wmiobject ?

L'information de $dqrwgpq après le dernier appel à la cmdlet est :
« . Get-WmiObject Win32_Process -Filter "ProcessId = '$(. Get-WmiObject Win32_Process -Filter "ProcessId = ' [System.Diagnostics.Process]::GetCurrentProcess() | . Select-Object -ExpandProperty ID'".ParentProcessId)'"" »
Voir image ci-contre :

```
xpandProperty ID', '$dQRWGpQ': ' . Get-WmiObject Win32_Process -Filter "ProcessId = \'$( . Get-WmiObject Win32_Process -Filter "Proces
sId = \' [System.Diagnostics.Process]::GetCurrentProcess() | . Select-Object -ExpandProperty ID\'".ParentProcessId)\'"', '$Wrm8EMP': '
```

c. Quel est le but de la cmdlet 1+1 ?

La commande cmdlet (1+1) décrit la fonction suivante :

```
. iex (function 1+1($b, $k) {
    $n = New-Object System.Collections.Generic.List[System.Object];
    for ($i = 0; $i -lt $b.Count; $i++) {
        $n.Add([char]($b[$i] -bxor $k[$i]))
    }
    return -join $n.ToArray();
})
```

Cette fonction récupère en argument 2 liste B et K correspondant à des chaînes de caractère, puis réalise le xor des deux.

d. Quel la cmdlet et les arguments utilisée dans le block catch après l'ASCII art ?

Après le ASCII ART une commande cmdlet écrivant dans le terminal est éxécuté :

```
catch {
    . Write-Output (Pikachu n est pas disponible en dynamax !!!)
}
```

Les arguments sont donc « Pikachu n'est pas disponible en dynamax !!! »

e. Quel cmdlet doit être stockée dans la variable $wrm8emp afin d'afficher Dracaufeu ?

La cmdlet qui doit être stockée dans la variable $wrm8emp afin d'afficher Dracaufeu est :

```
$Wrm8EMP = (.((1+1))@(3,0,8,23,4,88,41,16,1,94,16,12) . Get-WmiObject Win32_Process -Filter
"ProcessId = '$( . Get-WmiObject Win32_Process -Filter "ProcessId = ' [
System.Diagnostics.Process]::GetCurrentProcess() | . Select-Object -ExpandProperty
ID'".ParentProcessId)'".(ProcessName));
```

f. Quel est donc la condition pour que Dracaufau soit affiché ?

En étudiant la dernière commande cmdlet $WRM8EMP, on remarque que celle-ci récupère le nom du processus exécuté, ce qui correspondrait à « Pikachu » pour « pikachu.exe ». Mais on remarque également qu'avant, cette variable réalise la fonction(1+1) vue précédemment :

```
$Wrm8EMP = (.((1+1))@(3,0,8,23,4,88,41,16,1,94,16,12) . Get-WmiObject Win32_Process -Filter
"ProcessId = '$( . Get-WmiObject Win32_Process -Filter "ProcessId = ' [
System.Diagnostics.Process]::GetCurrentProcess() | . Select-Object -ExpandProperty
ID'".ParentProcessId)'".(ProcessName));
```

On en déduit alors qu'un xor est réalisé entre @(3,0,8,23,4,88,41,16,1,94,16,12) et le nom de l'exécutable.

g. Quel processus doit-on suivre pour invoquer Dracaufeu depuis le binaire originel ?

Pour invoquer Dracaufeu depuis le binaire originel, il faut simplement renommer le fichier (non compressé) pikachu.exe en dracaufeu.exe puis en passant en argument la clé « dynamax ». Clé présente ici :

```
v20 = 'xamanyd';
```

On obtient alors :


PS [redacted]> ./dracaufeu.exe dynamax
Loading...