Ecriture dans un fichier

Localisation: `0x00401D1C` **Fonction:** `VirtualProtect`
Type: Anti-debug **Sévérité:** Élevée

Code Assembleur

```
.text:00404898       mov     [esp+4Ch+Stream], 40h ; @ ; flNewProtect
.text:004048A0       mov     [esp+4Ch+pbDebuggerPresent], 19Eh ; dwSize
.text:004048A8       mov     [esp+4Ch+hProcess], offset byte_4786A0 ; lpAddress
.text:004048AF       call    ds:VirtualProtect
```

Analyse

La fonction `sub_4047A0` agit comme un *loader* : son rôle est d'utiliser une donnée "externe" pour reconstruire un morceau de programme. Premièrement la fonction verifie si un débugger est présent à l'aide de la fonction native `CheckRemoteDebuggerPresent`. Si un debugger est détecté, elle saute à l'adresse `loc_404800` (nous ne détaillons pas le comportement si un debugger est présent). Dans le cas contraire, la fonction va chercher directement sur le disque à la localisation suivante :

"C:\Users\lhs\AppData\Local\Temp\astiko.txt".

La fonction tente d'ouvrir ce fichier avec `fopen`. Si `fopen` échoue, alors la fonction saute à l'erreur "POUET", d'où la présence de cette chaîne de caractères dans les *strings* de l'exe. Ensuite, la fonction lit les 20 (14h) octets présents dans ce fichier texte. Nous avons découvert plus tôt que le fichier texte contient la chaîne de caractère suivante, servant de clé de déchiffrement:

pakbo-et-lombrik.fr

Ensuite le déchiffrement commence. La ligne `xor byte_4786A0[ecx], al` indique que le chiffré se trouve à l'adresse `0x4786A0`. La boucle de déchiffrement tourne 414 fois (`19Eh`). Cela nous indique que le chiffré est long de 414 octets.

Une fois les 414 octets déchiffrés, le programme fait un appel à `VirtualPotect` afin de changer les permissions d'une zone mémoire. L'appel à `VirtualProtect` est détaillé ci-dessus :

- lpAddress (0x4786A0) : l'adresse du début de la zone mémoire dont on veut changer les permissions. C'est une adresse statique dans le segment de donnée du programme ;
- dwSize (0x19E = 414 octets) : la taille de la région à protéger ;
- flNewProtect (0x40) : la nouvelle protection demandée correspond à `PAGE_EXECUTE_READWRITE`, ce qui signifie que la zone devient lisible, modifiable et exécutable ;
- pflOldProtect : un pointeur vers une variable locale sur la *stack* (`[esp+4Ch+flOldProtect]`) qui recevra l'ancienne valeur de protection.

Ensuite, nous avons extrait les 414 octets chiffrés à l'aide du script python ci-dessous. On obtient le code machine suivant

```
55                   push   rbp
89 e5                mov    rbp,rsp
83 ec 24             sub    rsp,0x24
8b 45 08             mov    eax,DWORD PTR [rbp+0x8]
```

```
88 45 dc              mov    BYTE PTR [rbp-0x24],al
c6 45 fb 01           mov    BYTE PTR [rbp-0x5],0x1
c6 45 fa 00           mov    BYTE PTR [rbp-0x6],0x0
c6 45 f9 01           mov    BYTE PTR [rbp-0x7],0x1
c7 45 f4 2a 00 00 00  mov    DWORD PTR [rbp-0xc],0x2a
0f b6 45 fb           movzx  eax,BYTE PTR [rbp-0x5]
84 c0                 test   al,al
74 0f                 je     0x35
0f b6 45 fa           movzx  eax,BYTE PTR [rbp-0x6]
84 c0                 test   al,al
74 07                 je     0x35
b8 01 00 00 00        mov    eax,0x1
eb 05                 jmp    0x3a
b8 00 00 00 00        mov    eax,0x0
88 45 f3              mov    BYTE PTR [rbp-0xd],al
0f b6 45 fb           movzx  eax,BYTE PTR [rbp-0x5]
84 c0                 test   al,al
75 08                 jne    0x52
0f b6 45 fa           movzx  eax,BYTE PTR [rbp-0x6]
84 c0                 test   al,al
74 07                 je     0x59
b8 01 00 00 00        mov    eax,0x1
eb 05                 jmp    0x5e
b8 00 00 00 00        mov    eax,0x0
88 45 f2              mov    BYTE PTR [rbp-0xe],al
0f b6 45 f9           movzx  eax,BYTE PTR [rbp-0x7]
83 f0 01              xor    eax,0x1
88 45 f1              mov    BYTE PTR [rbp-0xf],al
0f b6 55 f3           movzx  edx,BYTE PTR [rbp-0xd]
0f b6 45 f2           movzx  eax,BYTE PTR [rbp-0xe]
38 c2                 cmp    dl,al
0f 95 c0              setne  al
88 45 f0              mov    BYTE PTR [rbp-0x10],al
c7 45 fc 00 00 00 00  mov    DWORD PTR [rbp-0x4],0x0
eb 16                 jmp    0x9f
8b 45 fc              mov    eax,DWORD PTR [rbp-0x4]
83 e0 01              and    eax,0x1
85 c0                 test   eax,eax
0f 94 c0              sete   al
88 45 eb              mov    BYTE PTR [rbp-0x15],al
0f b6 45 eb           movzx  eax,BYTE PTR [rbp-0x15]
83 45 fc 01           add    DWORD PTR [rbp-0x4],0x1
83 7d fc 04           cmp    DWORD PTR [rbp-0x4],0x4
0f 9e c0              setle  al
84 c0                 test   al,al
75 df                 jne    0x89
8b 45 f4              mov    eax,DWORD PTR [rbp-0xc]
c1 e8 1f              shr    eax,0x1f
84 c0                 test   al,al
74 0c                 je     0xba
0f b6 45 dc           movzx  eax,BYTE PTR [rbp-0x24]
```

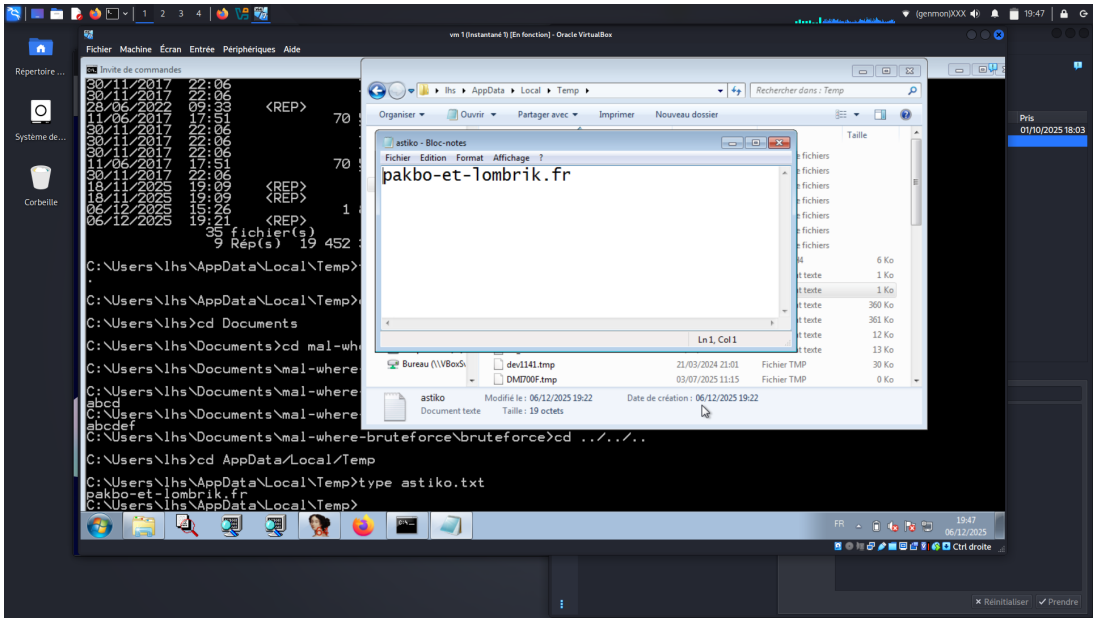
```
83 f0 01              xor    eax,0x1
e9 e5 00 00 00        jmp    0x1a4
8b 55 f4              mov    edx,DWORD PTR [rbp-0xc]
8b 45 f4              mov    eax,DWORD PTR [rbp-0xc]
0f af c2              imul   eax,edx
85 c0                 test   eax,eax
0f 9f c0              setg   al
88 45 ef              mov    BYTE PTR [rbp-0x11],al
0f b6 45 ef           movzx  eax,BYTE PTR [rbp-0x11]
83 f0 01              xor    eax,0x1
84 c0                 test   al,al
74 0a                 je     0xe6
b8 00 00 00 00        mov    eax,0x0
e9 bf 00 00 00        jmp    0x1a4
0f b6 45 dc           movzx  eax,BYTE PTR [rbp-0x24]
88 45 ee              mov    BYTE PTR [rbp-0x12],al
0f b6 45 ee           movzx  eax,BYTE PTR [rbp-0x12]
84 c0                 test   al,al
74 07                 je     0xfa
b8 01 00 00 00        mov    eax,0x1
eb 05                 jmp    0xff
b8 00 00 00 00        mov    eax,0x0
88 45 ee              mov    BYTE PTR [rbp-0x12],al
0f b6 45 ee           movzx  eax,BYTE PTR [rbp-0x12]
84 c0                 test   al,al
75 07                 jne    0x10f
b8 00 00 00 00        mov    eax,0x0
eb 05                 jmp    0x114
b8 01 00 00 00        mov    eax,0x1
88 45 ee              mov    BYTE PTR [rbp-0x12],al
0f b6 45 ee           movzx  eax,BYTE PTR [rbp-0x12]
88 45 ee              mov    BYTE PTR [rbp-0x12],al
0f b6 45 ee           movzx  eax,BYTE PTR [rbp-0x12]
88 45 ee              mov    BYTE PTR [rbp-0x12],al
90                    nop
90                    nop
90                    nop
90                    nop
90                    nop
90                    nop
90                    nop
90                    nop
90                    nop
90                    nop
0f b6 45 fb           movzx  eax,BYTE PTR [rbp-0x5]
84 c0                 test   al,al
74 12                 je     0x14b
0f b6 45 fb           movzx  eax,BYTE PTR [rbp-0x5]
83 f0 01              xor    eax,0x1
84 c0                 test   al,al
74 07                 je     0x14b
```

```
b8 01 00 00 00        mov    eax,0x1
eb 05                 jmp    0x150
b8 00 00 00 00        mov    eax,0x0
84 c0                 test   al,al
74 07                 je     0x157
b8 01 00 00 00        mov    eax,0x1
eb 48                 jmp    0x1a4
0f b6 45 dc           movzx  eax,BYTE PTR [rbp-0x24]
88 45 ed              mov    BYTE PTR [rbp-0x13],al
0f b6 45 ed           movzx  eax,BYTE PTR [rbp-0x13]
83 f0 01              xor    eax,0x1
88 45 ed              mov    BYTE PTR [rbp-0x13],al
0f b6 45 ed           movzx  eax,BYTE PTR [rbp-0x13]
83 f0 01              xor    eax,0x1
88 45 ed              mov    BYTE PTR [rbp-0x13],al
90                    nop
90                    nop
90                    nop
90                    nop
90                    nop
0f b6 45 dc           movzx  eax,BYTE PTR [rbp-0x24]
88 45 ec              mov    BYTE PTR [rbp-0x14],al
0f b6 45 ec           movzx  eax,BYTE PTR [rbp-0x14]
84 c0                 test   al,al
75 06                 jne    0x18e
80 7d dc 00           cmp    BYTE PTR [rbp-0x24],0x0
74 07                 je     0x195
b8 01 00 00 00        mov    eax,0x1
eb 05                 jmp    0x19a
b8 00 00 00 00        mov    eax,0x0
88 45 ec              mov    BYTE PTR [rbp-0x14],al
0f b6 45 dc           movzx  eax,BYTE PTR [rbp-0x24]
c9                    leave
c3                    ret
```

Analyse

Ce code ne fait rien, c'est un leurre

Screenshot




```
import struct
import os


FILENAME = "main.exe"
TARGET_VA = 0x4786A0
SIZE = 0x19E


def get_file_offset(pe_file, virtual_address):
    """
    Convertit une Virtual Address (VA) en File Offset (Raw Address)
    en analysant les sections du fichier PE.
    """
    try:

        pe_file.seek(0x3C)
        pe_header_offset = struct.unpack('<I', pe_file.read(4))[0]

        # Verification signature PE
        pe_file.seek(pe_header_offset)
        if pe_file.read(4) != b'PE\x00\x00':
            print("[-] Erreur : Ce n'est pas un fichier PE valide.")
            return None

        # Lecture du File Header (nombre de sections)
        pe_file.seek(pe_header_offset + 6)
        num_sections = struct.unpack('<H', pe_file.read(2))[0]

        # Lecture de l'Optional Header
        pe_file.seek(pe_header_offset + 24)
```

```
        magic = struct.unpack('<H', pe_file.read(2))[0]

        # Calcul de la taille de l'Optional Header
        pe_file.seek(pe_header_offset + 20)
        opt_header_size = struct.unpack('<H', pe_file.read(2))[0]

        # Recuperation de l'ImageBase
        pe_file.seek(pe_header_offset + 24)
        if magic == 0x10b:  # PE32 (32-bit)
            pe_file.seek(pe_header_offset + 24 + 28)
            image_base = struct.unpack('<I', pe_file.read(4))[0]
        elif magic == 0x20b:  # PE32+ (64-bit)
            pe_file.seek(pe_header_offset + 24 + 24)
            image_base = struct.unpack('<Q', pe_file.read(8))[0]
        else:
            print("[-] Format PE inconnu.")
            return None

        print(f"[i] Image Base detectee : {hex(image_base)}")

        # Calcul de la RVA (Relative Virtual Address)
        target_rva = virtual_address - image_base
        print(f"[i] RVA Cible : {hex(target_rva)}")

        # Debut de la table des sections
        section_table_offset = pe_header_offset + 24 + opt_header_size
        pe_file.seek(section_table_offset)

        # Parcours des sections pour trouver celle qui contient notre adresse
        for i in range(num_sections):
            # Structure Section Header
            pe_file.seek(section_table_offset + (i * 40))
            sec_name = pe_file.read(8).strip(b'\x00').decode(errors='ignore')
            v_size, v_addr, r_size, r_ptr = struct.unpack('<IIII', pe_file.read(16))

            # Verification si l'adresse est dans cette section
            if v_addr <= target_rva < (v_addr + max(v_size, r_size)):
                print(f"[+] Adresse trouvee dans la section : .{sec_name}")
                # Formule magique : Offset = RVA - VirtualAddress + PointerToRawData
                file_offset = target_rva - v_addr + r_ptr
                return file_offset

        print("[-] Adresse introuvable dans les sections.")
        return None

    except Exception as e:
        print(f"[-] Erreur lors de l'analyse : {e}")
        return None


def extract_data():
```

```
    global FILENAME
    with open(FILENAME, 'rb') as f:
        print(f"[*] Analyse de {FILENAME}...")
        offset = get_file_offset(f, TARGET_VA)

        if offset is not None:
            print(f"[+] Offset fichier calcule : {hex(offset)}")

            f.seek(offset)
            data = f.read(SIZE)

            if len(data) == SIZE:
                print(f"[+] {len(data)} octets extraits avec succes !")

                # Sauvegarde dans un fichier blob
                output_file = "encrypted_blob.bin"
                with open(output_file, "wb") as out:
                    out.write(data)

                print(f"[i] Donnees sauvegardees dans '{output_file}'")


            else:
                print("[-] Erreur : Lecture incomplete.")


if __name__ == "__main__":
    extract_data()
```