

# Rendu - Introduction au reverse-engineering et mécanismes de sécurité Windows

Tristan Pinaudeau

## 1. Ordre de mission

### 1.1 Préliminaires

Pour analyser préalablement le fichier au format PE on peut utiliser la suite d'outils pev qui nous fournis un ensemble de commandes pour manipuler l'exécutable sous linux.

Ainsi on peut facilement extraire le *timestamp* de l'en-tête du binaire:

```
> readpe apt-secure-up.exe | grep -i stamp
Date/time stamp:          1611249405 (Thu, 21 Jan 2021 17:16:45 UTC)
```

De la même façon, le nombre de sections:

```
> readpe apt-secure-up.exe | grep -i section
Number of sections:       6
```

Dans ces six sections, affichons celle qui est executables:

```
> readpe apt-secure-up.exe | sed -n '/Sections/,//p' | grep -i -e name -e execute
Name:                      .text
                           IMAGE_SCN_MEM_EXECUTE
Name:                      .rdata
Name:                      .data
Name:                      .pdata
Name:                      .rsrc
Name:                      .reloc
```

En analysant les fonctions importées par le binaire, on constate d'abord que celles-ci proviennent de 11 bibliothèques différentes. De part leurs nom celui des fonctions utilisés, nous pouvons connaître un peu mieux le fonctionnement du programme:

- Il utilisera des fonctionnalités du noyau (notamment relatif à la manipulation de processus)
- Il fera appel à des algorithmes cryptographiques (AES, RSA et SHA256)
- Il manipulera un certain nombre d'entrée/sortie (notamment liée au réseau)

```
> readpe -i apt-secure-up.exe | grep '^[^ ]'
```

Imported functions

KERNEL32.dll

libcrypto-1\_1-x64.dll

CRYPT32.dll

Cabinet.dll

WININET.dll

VCRUNTIME140.dll

api-ms-win-crt-stdio-l1-1-0.dll

api-ms-win-crt-heap-l1-1-0.dll

api-ms-win-crt-runtime-l1-1-0.dll

api-ms-win-crt-math-l1-1-0.dll

api-ms-win-crt-locale-l1-1-0.dll

La bibliothèque `Cabinet.dll` exporte ses fonctions au moyens d'une table d'ordinals:

```
> readpe -i apt-secure-up.exe | sed -n '/Cabinet.dll/,/^[^ ]/p'
```

Cabinet.dll

30

35

33

Cette librairie se trouve sous le chemin `C:\Windows\System32\cabinet.dll`. On y trouve les fonctions correspondantes que l'on peut lister:

```
> dumpbin.exe /exports cabinet.dll
```

...

ordinal	hint	RVA	name
35	0	000040B0	CloseCompressor
45	1	00004200	CloseDecompressor
33	2	00004070	Compress
30	3	000040C0	CreateCompressor
40	4	000041F0	CreateDecompressor

...

On constate que la librairie `cabinet` permet surtout l'utilisation de fonctions de compression. Le programme que l'on analyse ne semble se soucier que des méthodes de compression et non celles de décompression. On peut en déduire qu'il cherchera surtout à extraire de la donnée sur le réseau plutôt qu'en recevoir.

## 1.2 Reverse-engineering

La fonction `starter01` se contente de calculer la taille de la chaîne de caractère donnée en paramètre:

```
int length(char* param) {
    char* copy = param;
    while (*copy != '\0') copy++;
    return copy - param;
}
```

La fonction `starter02` permet de comparer deux chaînes de caractères. Elle retournera `0` lorsque les chaînes sont identiques. Sinon, elle compare les caractères un à un et au premier caractère différent, elle retourne `-1` si le caractère du premier paramètre est inférieur à celui du deuxième paramètre, et `1` dans le cas contraire.

```
int test(char* param_1, char* param_2) {
    char* c1 = param_1;
    char* c2 = param_2;
    while (1) {
        if (*c1 < *c2) return -1;
        else if (*c1 > *c2) return 1;
        else if (*c1 == '\0') return 0;
        c1++;
        c2++;
    }
}
```

En cherchant les références à cette fonction `starter02` dans ghidra on se rend compte qu'elles sont toutes présentes dans une fonction qui semble traiter les arguments du programme.

En excluant les trolls qui se cachent dans le code, l'analyse de cette fonction nous donne les informations suivantes:

- le programme attend 3 arguments
- le premier argument doit être `--password` ou `-p`
- le deuxième argument doit être un mot de passe de 9 caractères ou plus
- il ne semble pas y avoir de restrictions spécifiques sur le dernier argument

### 1.3 Analyse de la fonction `apt_func_001`

La fonction prend en entrée l'adresse d'une chaîne de caractères contenant le nom d'un fichier de clé publique au format PEM. Le deuxième argument est un pointeur vers une structure propre à la librairie `libcrypto.dll`. Le prototype est donc le suivant:

```
void apt_func_001(char *file_name, EVP_PKEY *pub_key);
```

### 1.4 Analyse de la fonction `apt_func_002`

La fonction utilise les fonctions de compression vues plus tôt dans la librairie `cabinet.dll`

**1.5 Analyse de la fonction apt\_func\_003**

**1.6 Analyse du chiffrement**

**1.7 data exfiltration**

**2. Strike back**

**3. Exploit in the wild**

## **Sources**

<https://docs.microsoft.com/en-us/windows/win32/debug/pe-format> <https://www.blackarch.org/tools.html>

<https://docs.microsoft.com/en-us/cpp/build/exporting-from-a-dll?view=msvc-160>