GATEWATCHER

ANALYSE DE MALWARE

Packer NSIS

Le programme d'installation NSIS (Nullsoft Scriptable Install System) est de plus en plus utilisé par les acteurs malveillants comme première étape d'unpacking. Comme il s'agit d'un logiciel libre (https://nsis.sourceforge.io/Download), il constitue une base pratique que les auteurs de logiciels malveillants peuvent utiliser et éventuellement modifier. La Purple Team Gatewatcher a analysé cette méthode de packing, nous allons la décrire en prenant le fichier d'exemple suivant :

SHA256:3807df31a0db07c4ea7b4e06c0f8c2db76e3c84071319d1e1b356c62f3872512 obtenu le 05/11/2022.

Le processus global d'unpacking de cet échantillon est le suivant :

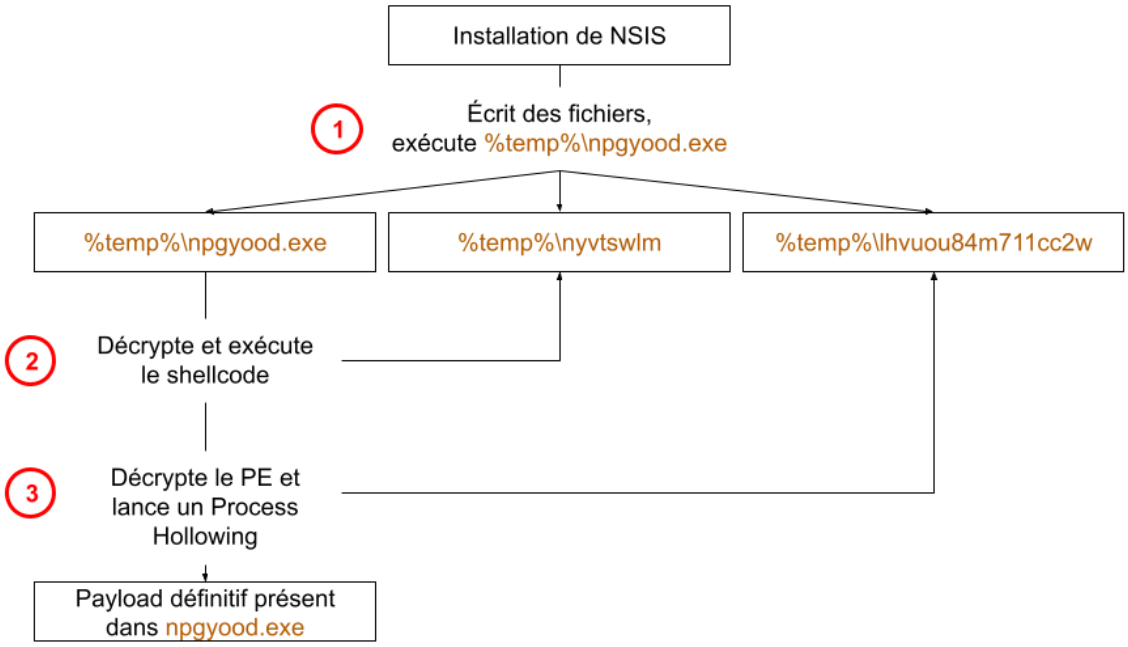

Figure 1 : Procédure d'unpacking

1. L'installateur NSIS est utilisé pour déposer 3 fichiers dans le dossier %temp% : un exécutable PE, et d'autres fichiers non identifiés. Il exécute ensuite le PE avant de se terminer.
2. Ce nouveau processus ouvre simplement l'un des autres fichiers déposés, le décrypte et saute sur son contenu (c'est un shellcode).
3. Le shellcode ouvre le troisième fichier, le décrypte, et exécute un process hollowing sur une copie de son propre processus avec la charge utile décryptée (un fichier PE).

Chacune de ces étapes sera analysée plus en détail dans cet article.

1 Installateur NSIS

1.1 Fichiers utilisant la compression BZIP2

L'échantillon que nous avons analysé utilise l'algorithme BZIP2 pour compresser ses données. Le fichier NSIS suit cette structure :

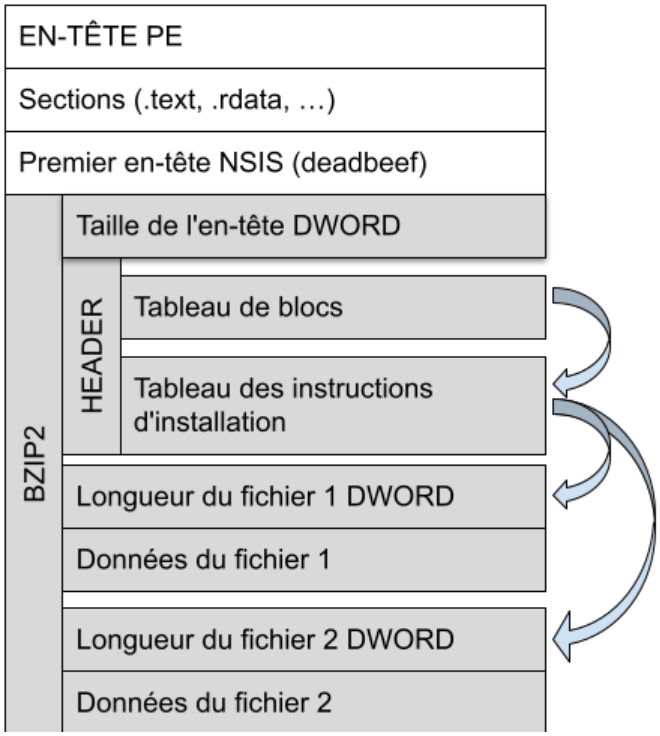

Figure 2: structure du fichier NSIS, lors de l'utilisation de BZIP2

Les données relatives à l'installation sont placées hors des sections PE, à la fin du fichier. Elles ne sont pas présentes dans la mémoire du processus.

Pour récupérer ce contenu, l'installateur commence par ouvrir son propre fichier, et recherche une structure d'en-tête (technique d'egg hunting):

- Un flag DWORD (avec une valeur inférieure à 16),
- Un DWORD: `0xdeadbeef`,
- La chaîne ASCII `NullSoftInst`,
- 2 DWORD: la longueur de l'en-tête, et la longueur des données suivantes.

Cet en-tête est suivi de données compressées par une version modifiée de BZIP2 (la taille de certains blocs a été modifiée par rapport à l'implémentation officielle, mais il s'agit globalement du même algorithme).

La décompression est progressive (toutes les données ne sont pas décompressées en une seule fois), et les données décompressées sont enregistrées dans un fichier aléatoire dans %temp% (utilisé comme un tableau d'octets à longueur variable).

Toutes les données du fichier sont compressées dans un seul flux, et chaque taille de champ (en-tête, longueur du fichier, ...) est stockée comme un DWORD à l'intérieur du flux compressé (ce qui est une différence majeure avec LZMA, comme nous le verrons plus tard).

Les données compressées commencent par un en-tête, avec un tableau de blocs. Chaque bloc pointe vers des données (des chaînes de caractères par exemple), et l'un d'entre eux pointe vers un tableau d'instructions d'installation, chacune d'entre elles étant décrite par un premier DWORD représentant le type d'instruction, et 6 autres DWORD utilisés comme paramètre, dont la signification varie en fonction de l'instruction (ce qui fait que la taille totale d'une entrée d'instruction est 0x1c).

Il existe environ 60 instructions, selon les options de compilation. Un résultat peut être stocké par une instruction pour être utilisé par la suivante, comme une forme de langage de programmation simpliste basé sur une pile.

Dans notre exemple, le numéro de l'instruction pour écrire un fichier est 0x14, l'offset de la structure de données du fichier dans le BZIP2 décompressé étant placé dans le deuxième paramètre (sur les 6). La structure de données du fichier est un simple DWORD représentant la taille du fichier, suivi du fichier lui-même.

L'algorithme de compression LZMA peut être utilisé à la place de BZIP2, avec exactement la même structure. Nous avons également rencontré des échantillons utilisant LZMA avec une structure différente.

1.2 Fichiers utilisant la compression LZMA par blocs

Certains échantillons NSIS utilisent l'algorithme de compression LZMA, avec une structure différente comme indiqué ci-dessous :

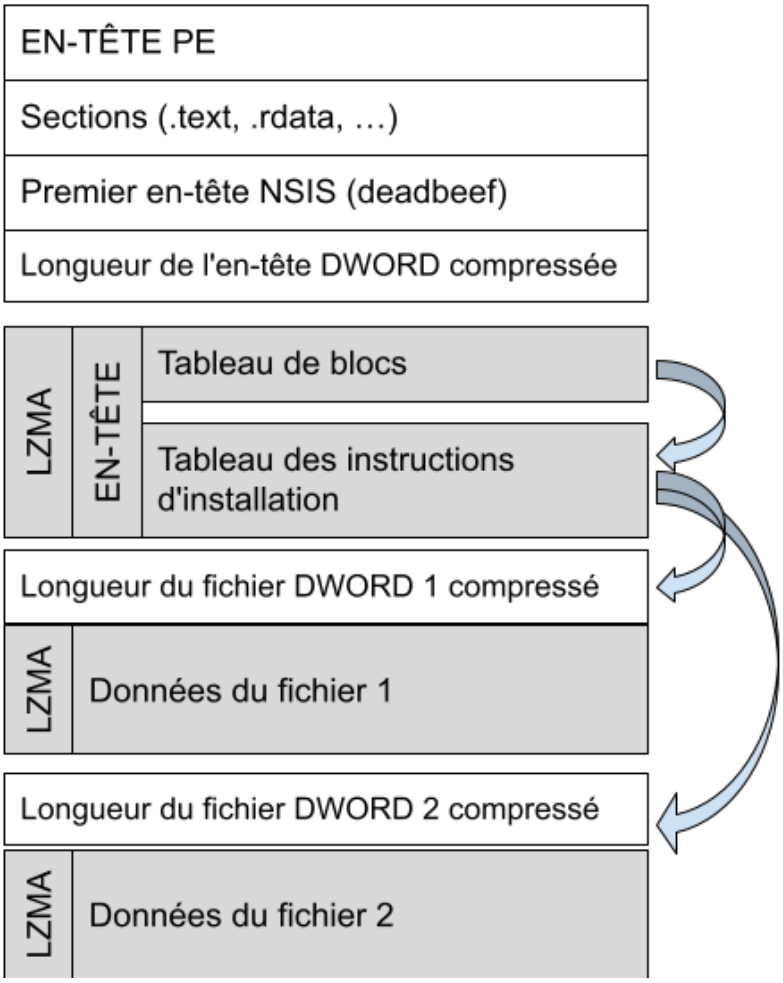

Figure 3 : Structure du fichier lors de l'utilisation de LZMA

La structure générale lors de l'utilisation de LZMA est la même, mais chaque champ commence par un DWORD en clair, indiquant la taille des données brutes à utiliser pour la décompression LZMA (ce DWORD est un champ de bits, et doit être xoré avec 0x7fffffff pour obtenir la taille réelle). Chaque champ du fichier est compressé indépendamment.

2 Fichiers déposés

L'échantillon analysé dépose 3 fichiers dans %temp%, l'un d'entre eux étant un exécutable PE lancé à la fin de l'installation. Le reste du processus de décompression se déroule uniquement en mémoire, aucun autre contenu n'est écrit sur le système de fichiers.

L'exécutable PE ouvre simplement et décrypte (une simple boucle de type XOR) l'un des deux autres fichiers déposés avant de sauter sur son contenu :

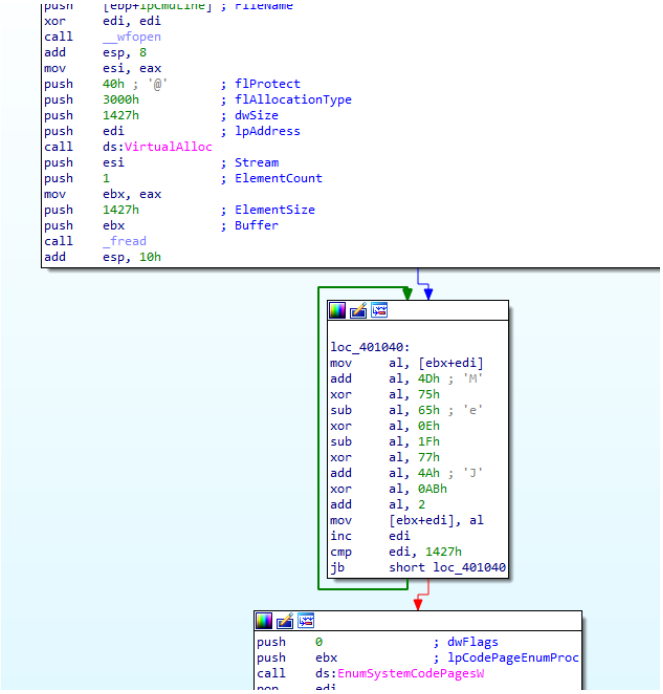

Figure 4: seconde étape d'unpacking

Ce shellcode obtient l'adresse de kernel32.dll à partir du PEB, et utilise un import par hash spécifique pour obtenir plusieurs autres fonctions:

```
call    RE_get_kernel32_dll
mov     [ebp+kernel32_dll], eax
mov     edx, 7FC01DAEh
mov     ecx, [ebp+kernel32_dll]
call    RE_custom_GetProcAddress ; GetTempPathW
mov     [ebp+GetTempPathW], eax
mov     edx, 0FF7F721Ah
mov     ecx, [ebp+kernel32_dll]
call    RE_custom_GetProcAddress ; GetModuleFileNameW
mov     [ebp+GetModuleFileNameW], eax
```

Figure 5: Appel de l'import par hash

Il procède ensuite à l'ouverture du troisième et dernier fichier déposé, le décrypte avec une autre (très longue) boucle de type XOR:

Figure 6 : vue partielle de la fonction de déchiffrement, basée sur des opérations ADD, OR, SUB, XOR, NOT et NEG successives.

Enfin, le shellcode redémarre son exécutable de processus, et effectue un process hollowing sur sa copie, en chargeant le PE décrypté en mémoire.

La charge utile finale de cet échantillon a été détectée comme étant une version commune de Loki.

3 Hunting

Nous avons trouvé de nombreux autres échantillons utilisant NSIS comme première étape de packing, la méthode de compression LZMA par blocs étant la plus utilisée. Les fichiers déposés sont dans de nombreux cas très similaires à ceux que nous avons présentés ci-dessus. La routine de déchiffrement exacte varie :

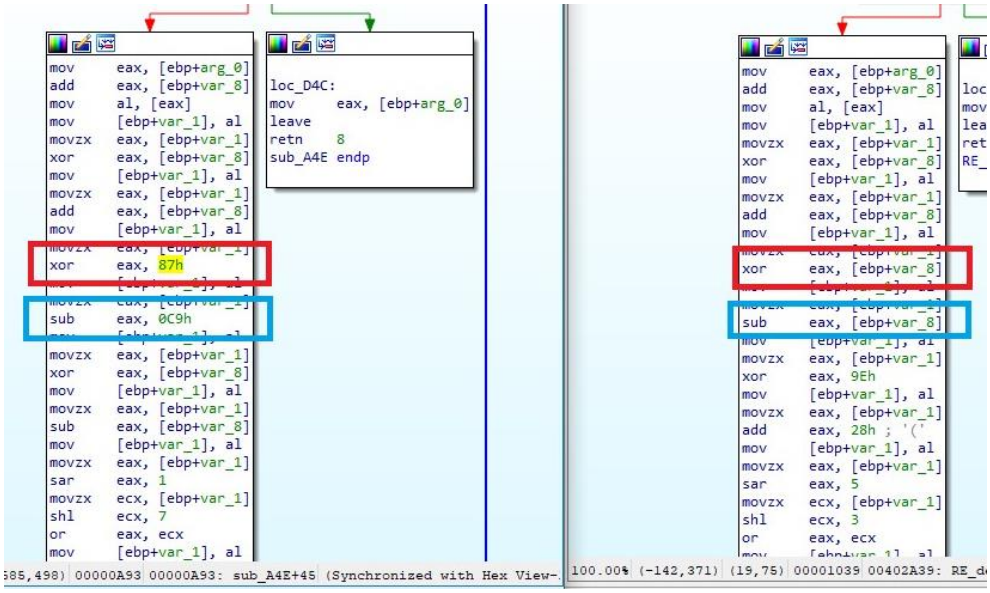

Figure 7 : 2 fonction de déchiffrement présente dans le shellcode en étape 3.

Dans l'exemple ci-dessus, le déchiffrement suit le même flux, mais les opérations (XOR, ADD, etc ...) et la valeur utilisée (soit la position actuelle, soit une constante) sont différentes : elle est probablement générée aléatoirement.


GATEWATCHER