Analyse du binaire « la_meuh.exe »

Partie 1 : Présentons le sujet

Pour cette analyse, je vais m'occuper de mon propre programme nommé « la_meuh », qui est un automatiseur de mises à jour de programmes pour Microsoft Windows.

Le programme appelle juste la commande « winget upgrade —all » en cmd caché, et automatise tout le processus de manière la plus user friendly possible et sans besoin d'UAC, car bon, utiliser cmd pour un utilisateur normal n'est pas simple, alors le programme rends tout simple et automatisé.

Pourquoi je fait cette analyse ?

- La meuh à été flaggé injustement comme trojan sur virustotal, sans doute car le process tree de ce binaire est « la_meuh.exe > cmd.exe > winget.exe », et les règles de détection des AV étant tellement arbitraires, et ne verifiant pas quelles commandes sont tapées dans la cmd (à savoir ici « winget upgrade —all »), elle à été injustement détectée. Cette méthode de détection est d'ailleurs surprenante, certains AV ne le font pas de manière intelligente.

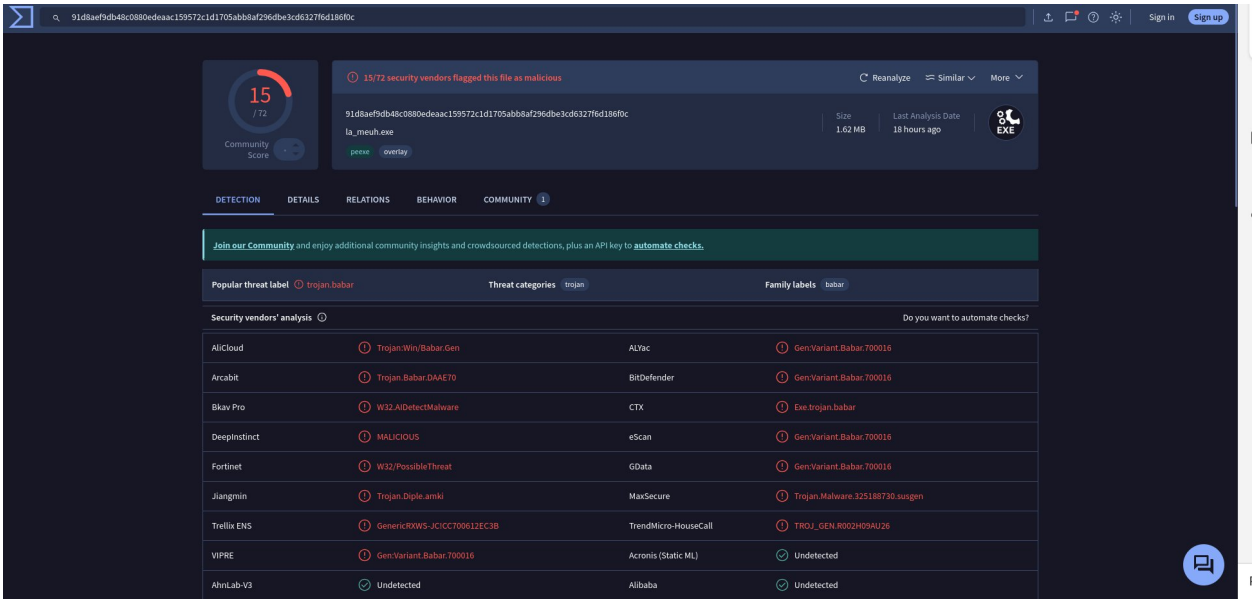
- Je vais donc analyser manuellement le binaire, pour prouver par les faits que ceci est un faux positif, et vraiment c'est dommage d'en arriver là car c'est juste un petit programme tout simple et légitime pour compenser ce que Windows ne sait pas faire nativement : mettre à jour les programmes.

Pour cette analyse je serai armé de ma machine Kali Linux et ses outils natifs, de Ghidra et de divers sites de threat intelligence et analyse de malware.

Dans un premier temps nous allons nous procurer les hashs du fichier :

```
┌──(kali㉿kali)-[~/analyse de binaires]
└─$ sha256sum "la_meuh.exe"
91d8aef9db48c0880edeaac159572c1d1705abb8af296dbe3cd6327f6d186f0c  la_meuh.exe
```

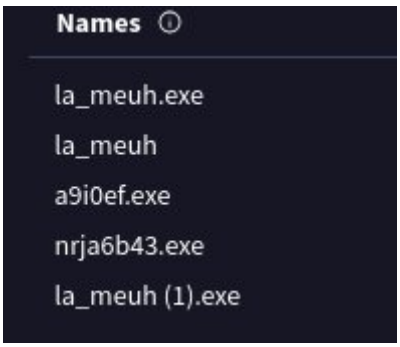
Une fois le hash récupéré, je lance une analyse virustotal pour l'exemple :



Comme on le voit ici, le programme est détecté comme trojan babar (principalement).

La plupart des AV qui le flag sont des AV très peu connus, ce qui explique sûrement le problème.

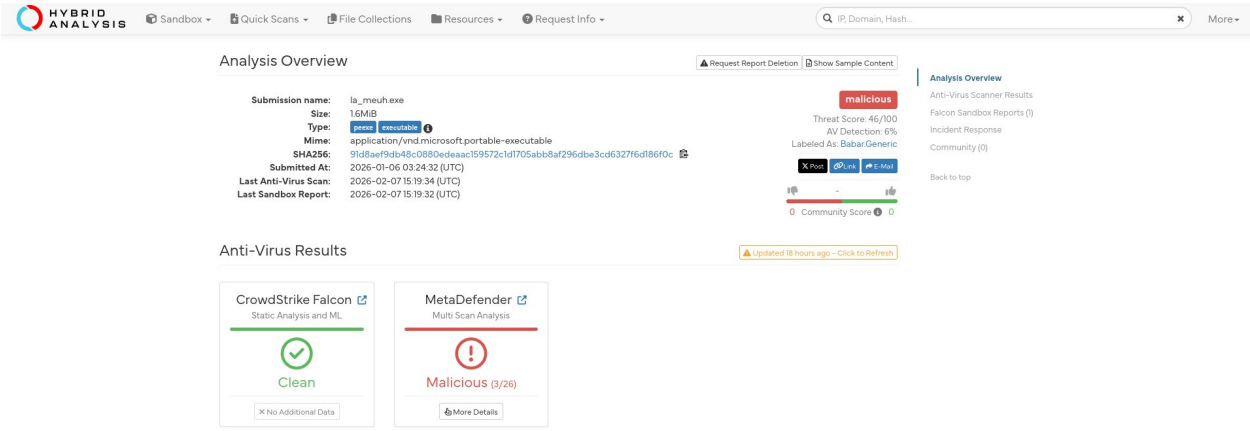
Si on regarde dans les noms du fichier, cela est surprenant, car il à des noms que je ne connais pas et je n'ai jamais appelé le binaire ainsi, il serait alors confondu avec des vrais logiciels malveillants ?



En faisant des recherches sur les deux noms que nous ne connaissons pas, il s'agit effectivement de malwares réels.

Si vous voulez avoir plus d'informations sur le résultat virustotal de la meuh, voici le lien : https://www.virustotal.com/gui/file/91d8aef9db48c0880edeaac159572c1d1705abb8af296dbe3cd6327f6d186f0c/detection

Passons maintenant à Hybrid Analysis :



Hybrid analysis flag lui aussi la_meuh comme étant malveillante, en effet, le fonctionnement du logiciel à savoir de lancer « winget upgrade —all » en cmd caché pose bien des problèmes pour sa survie.

Avant de partir au combat, je laisse ici le lien du projet de la meuh sur github, cela effacera tout doute pour les gens qui s'y connaissent en développement et en programmes malveillants :

https://github.com/spellskite-coding/la_meuh

Partie 2 : Analyse statique

Pour cette partie d'analyse statique, je vais commencer par lancer la commande « detect-it-easy » pour avoir plus d'informations sur le binaire :

```
┌──(kali㉿kali)-[~/analyse de binaires]
└─$ diec la_meuh.exe

[!] Heuristic scan is disabled. Use '--heuristicscan' to enable
PE32
    Linker: GNU Linker ld (GNU Binutils)(2.28)[GUI32]
    Compiler: MinGW(GCC: (GNU) 6.3.0)
```

Ce que cette sortie prouve :

- Binaire Windows PE32 standard
- Compilé avec MinGW GCC 6.3.0
- Pas packé
- Pas obfusqué
- Pas de protecteur
- Pas de runtime exotique (Go/Rust/.NET)

Ensuite il est temps de regarder quelles DLLs sont appelées par le programme :

```
┌──(kali㉿kali)-[~/analyse de binaires]
└─$ objdump -p la_meuh.exe | grep DLL

vma:            Hint    Temps      Avant    DLL       Premier
        Nom DLL: COMCTL32.DLL
        Nom DLL: GDI32.dll
        Nom DLL: KERNEL32.dll
        Nom DLL: msvcrt.dll
        Nom DLL: msvcrt.dll
        Nom DLL: USER32.dll
```

Puis un check de strings suspects :

```
┌──(kali㉿kali)-[~/analyse de binaires]
└─$ objdump -p la_meuh.exe | grep -E "CreateRemoteThread|WriteProcessMemory|VirtualAllocEx|WinInet|WinHttp|WS2_32|Internet"
```

Rien ici.

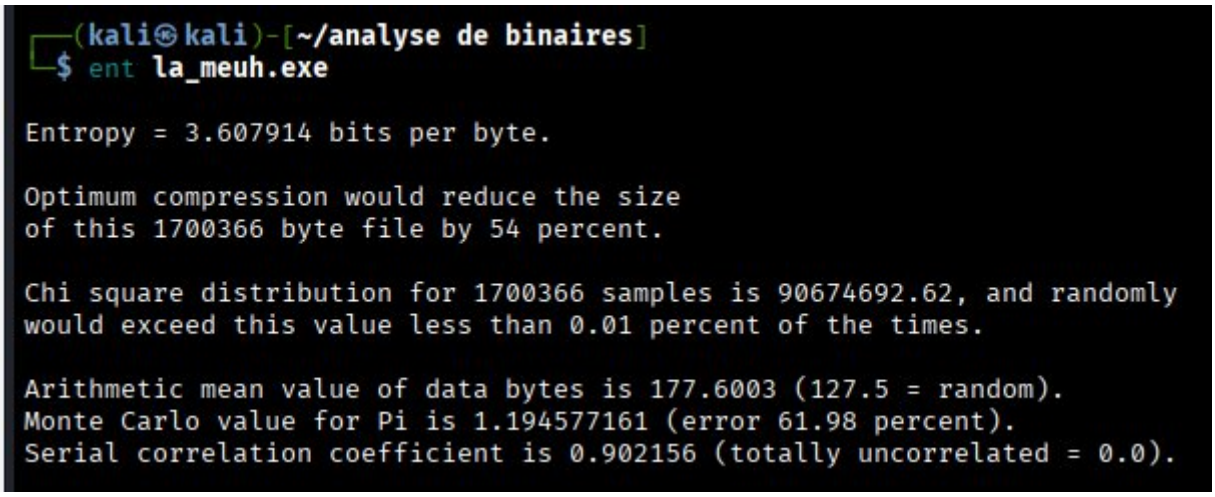
```
┌──(kali㉿kali)-[~/analyse de binaires]
└─$ strings -n 6 la_meuh.exe | grep -Ei "http|https|ftp|socket|connect|cmd|powershell|reg"

__register_frame_info
__deregister_frame_info
_Jv_RegisterClasses
RegisterClassA
X86_TUNE_PARTIAL_REG_DEPENDENCY
X86_TUNE_SSE_PARTIAL_REG_DEPENDENCY
X86_TUNE_SSE_SPLIT_REGS
X86_TUNE_PARTIAL_FLAG_REG_STALL
X86_TUNE_GENERAL_REGS_SSE_SPILL
X86_TUNE_PARTIAL_REG_STALL
X86_TUNE_PROMOTE_HI_REGS
X86_TUNE_PROMOTE_QI_REGS
reg_class
NO_REGS
AD_REGS
CLOBBERED_REGS
Q_REGS
NON_Q_REGS
INDEX_REGS
LEGACY_REGS
GENERAL_REGS
FP_TOP_REG
FP_SECOND_REG
FLOAT_REGS
SSE_FIRST_REG
NO_REX_SSE_REGS
SSE_REGS
EVEX_SSE_REGS
BND_REGS
ALL_SSE_REGS
MMX_REGS
FP_TOP_SSE_REGS
FP_SECOND_SSE_REGS
FLOAT_SSE_REGS
FLOAT_INT_REGS
INT_SSE_REGS
FLOAT_INT_SSE_REGS
MASK_EVEX_REGS
MASK_REGS
ALL_REGS
LIM_REG_CLASSES
dbx_register_map
dbx64_register_map
svr4_dbx_register_map
x86_64_ms_sysv_extra_clobbered_registers
regclass_map
_deregister_frame_fn
___gcc_register_frame
___gcc_deregister_frame
___tlregdtor
_register_frame_ctor
__imp__RegisterClassA@4
___register_frame_info
.weak.___register_frame_info.___EH_FRAME_BEGIN__
.weak.___deregister_frame_info.___EH_FRAME_BEGIN__
.weak.__Jv_RegisterClasses.___EH_FRAME_BEGIN__
__Jv_RegisterClasses
___deregister_frame_info
_RegisterClassA@4
```

Ici la sortie est classique, rien de suspicieux, il y a les métadonnées du compilateur GCC, l'API de GUI classique de la création de fenêtres, les runtimes GCC et gestion des exceptions. Rien de louche, pas de clés de registre Windows modifiées, pas de commandes systèmes ou d'actions de persistance.

Voici ce que prouvent les deux sorties :

L'analyse statique du binaire la_meuh.exe montre l'absence totale d'APIs réseau, d'injection de code, de persistance ou de comportement système sensible. Les chaînes détectées correspondent exclusivement au runtime GCC et aux mécanismes standards de l'API graphique Windows. Aucun indicateur de compromission n'est présent.

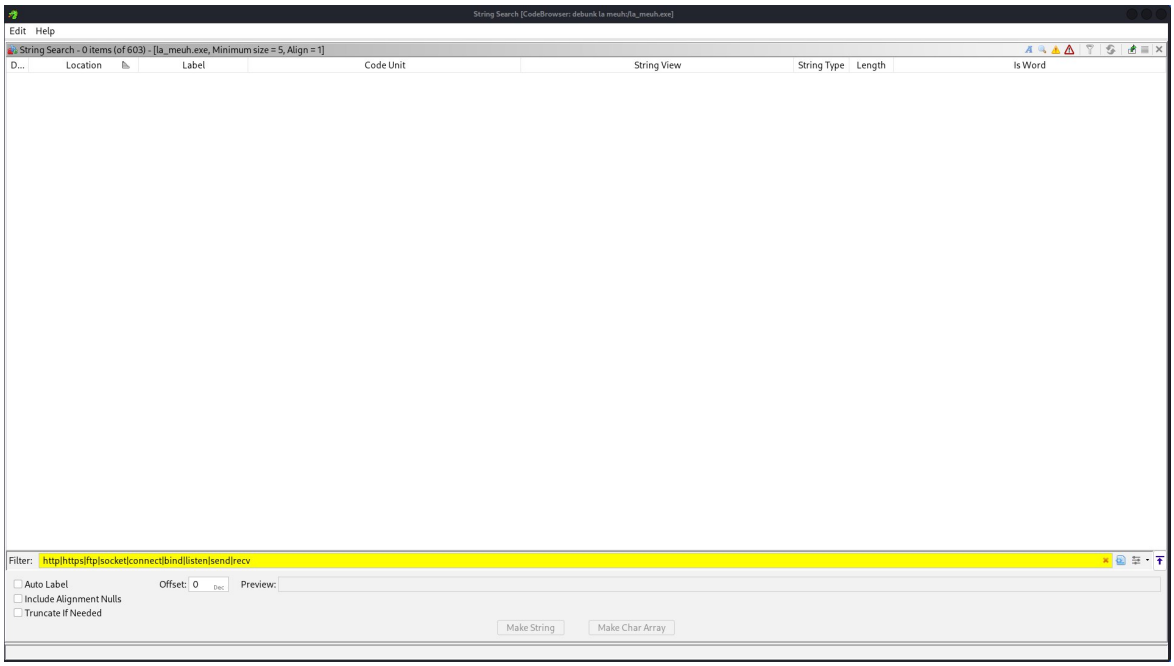
Analyse entropique du programme :



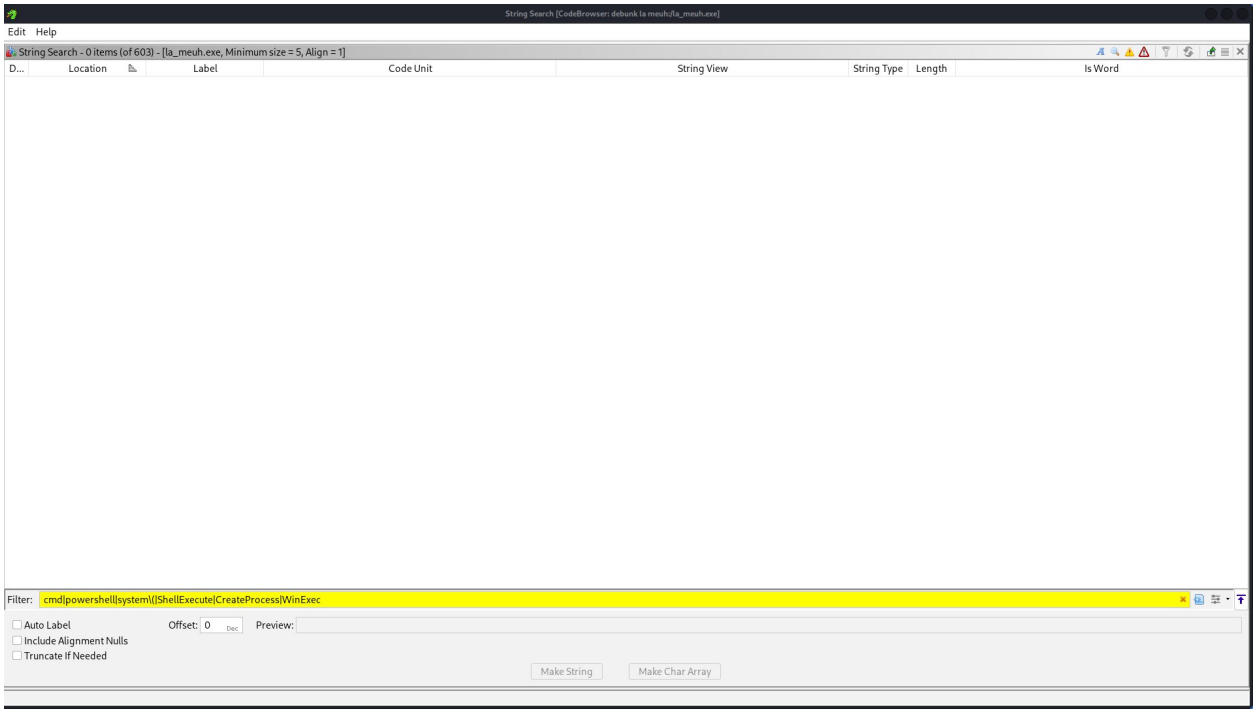
L'analyse entropique du binaire la_meuh.exe révèle une entropie moyenne de 3.61 bits par octet, valeur caractéristique d'un exécutable non packé et non chiffré. Cette métrique exclut l'utilisation de packers, de techniques de dissimulation ou de chiffrement, incompatibles avec les pratiques observées dans les malwares modernes. Le binaire contient des données hautement structurées et redondantes, cohérentes avec un programme compilé standard.

Partie 3 : Analyse du binaire avec Ghidra

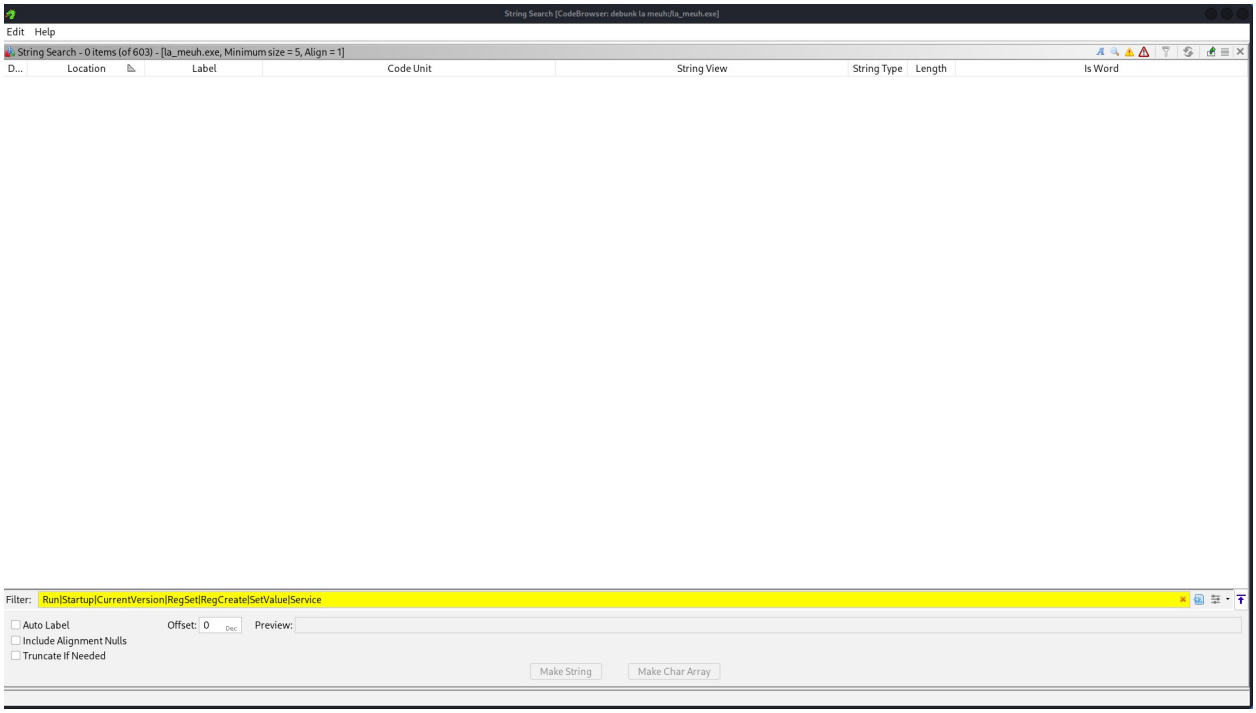
Je décompile le programme puis fait une recherche par strings, et je cherche en premier lieu tout ce qui est en lien avec le réseau, et comme attendu je ne trouve rien :



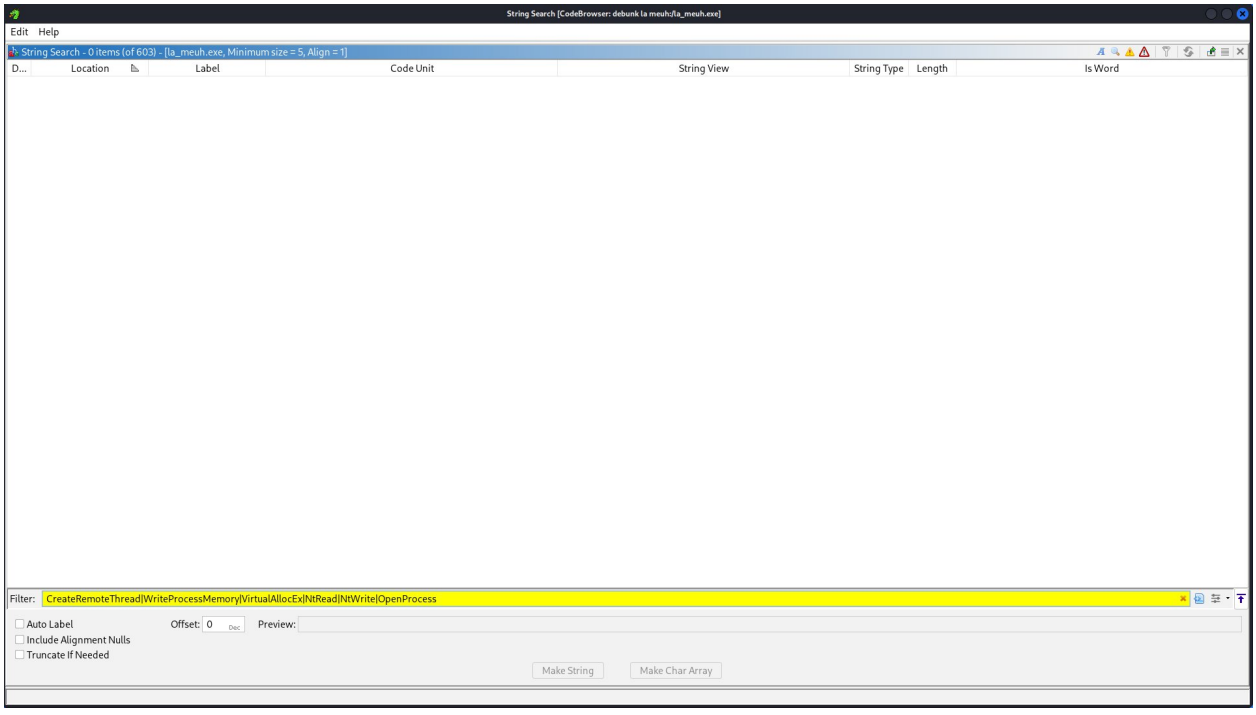
Ensuite une recherche de commandes sytème :



Puis une recherche de persistance :

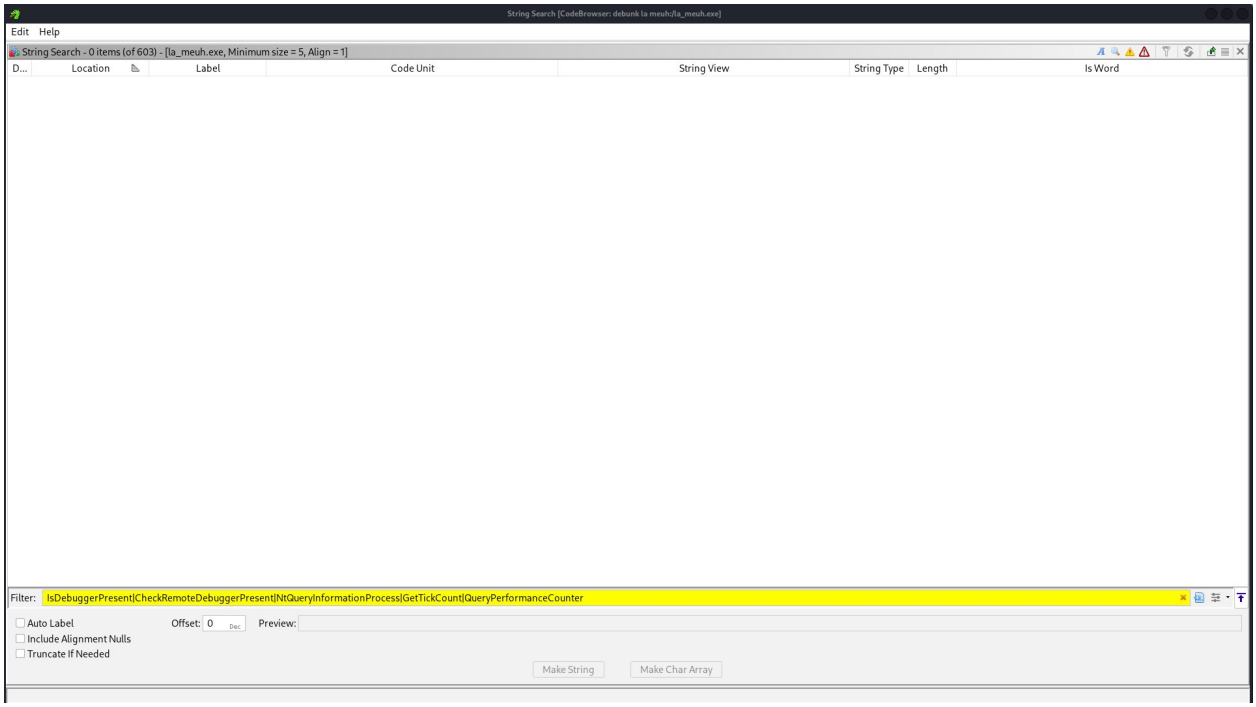


Injection/manipulation de mémoire peut être ?



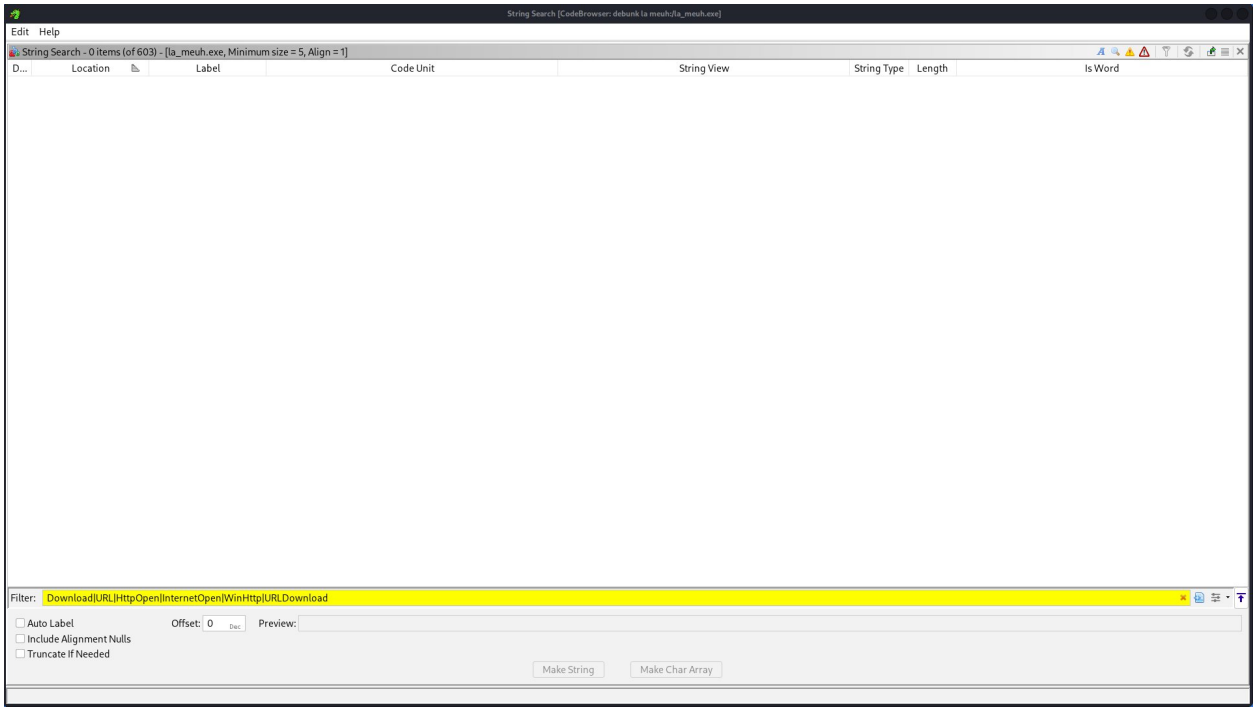
Toujours rien :)

Un check anti-debugging et anti VM ?



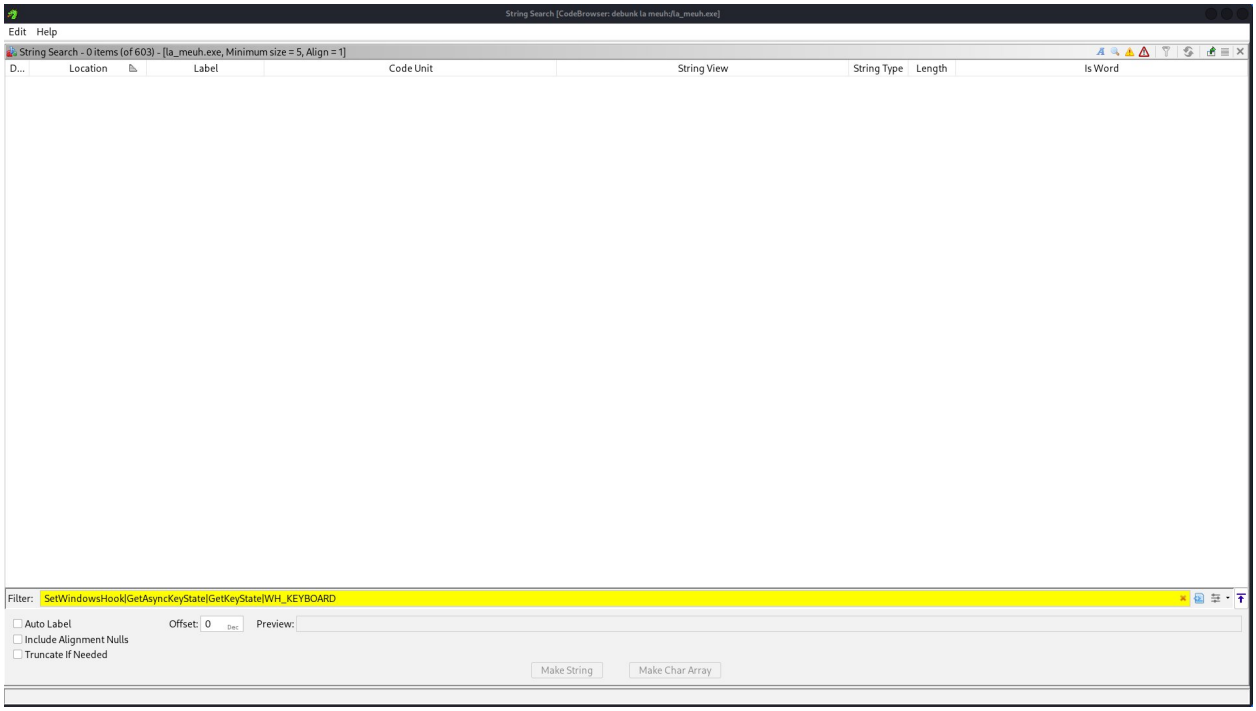
Rien ici non plus :)

Téléchargement d'autres binaires ou autre ?



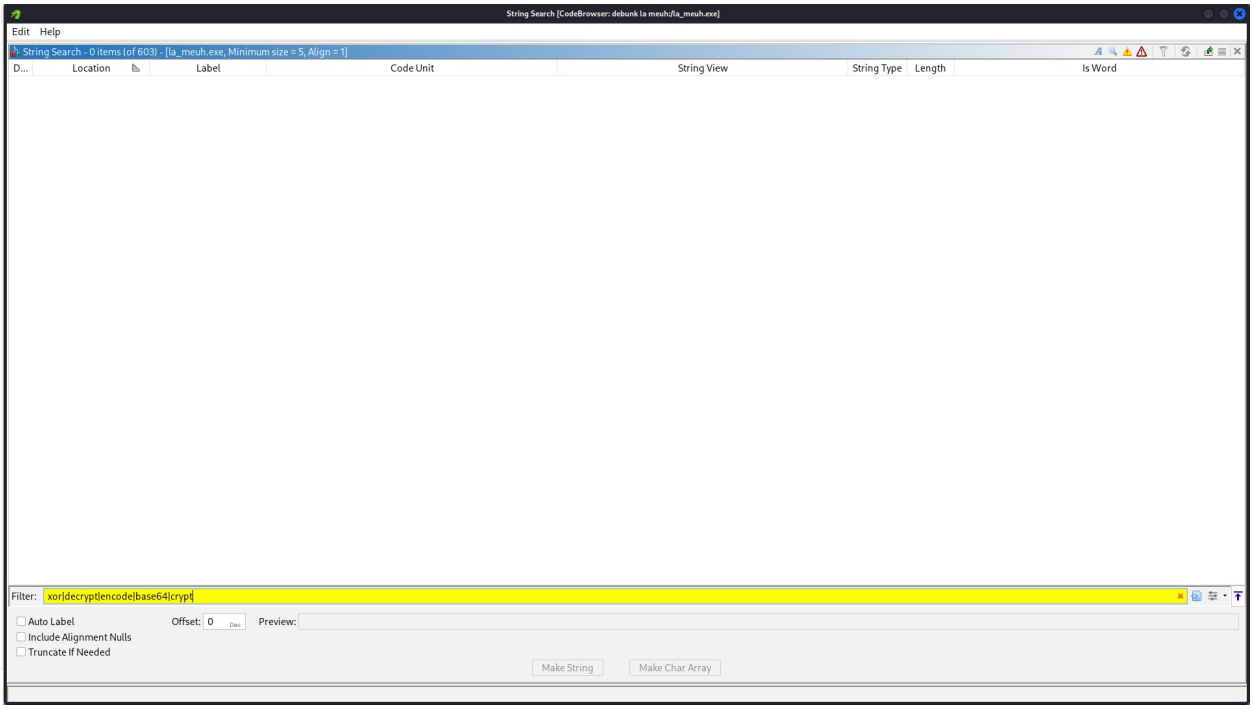
Rien du tout comme attendu.

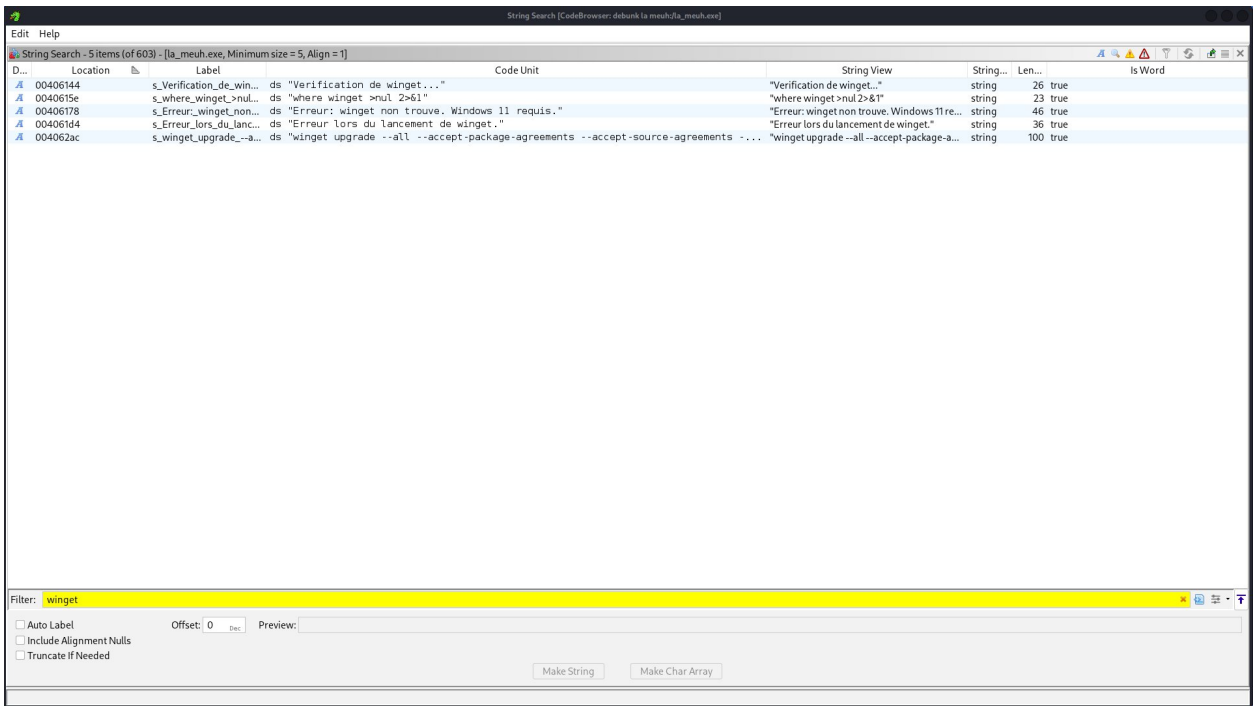
Une recherche de keylogging :



Toujours rien !

Une recherche d'obfuscation pour la route :





En recherchant explicitement le mot-clé winget, on identifie les commandes liées à l'exécution de winget que le programme lance, ce qui est nécessaire pour automatiser la mise à jour des applications sous Windows.

Les autres recherches effectuées dans Ghidra visaient à détecter des comportements typiquement associés aux malwares, tels que l'ouverture d'un interpréteur de commandes ou l'exécution de commandes arbitraires et malveillantes. Leur absence confirme que le programme n'utilise pas ces mécanismes.

L'analyse montre que le programme lance winget via cmd en utilisant l'API Windows standard (CreateProcess), et uniquement winget, sans passer par un shell interactif ni exécuter d'autres commandes.

Conclusion de l'analyse :

Le programme est totalement sain et légitime, et cette recherche manuelle dans le binaire décompilé par Ghidra le prouve, tout comme cette analyse statique.

En vue des faits établis, cela serait ridicule de faire ensuite une analyse dynamique, on verrais juste le process tree mentionné tout en haut du document, et ce serait winget qui ferait les appels réseau pour télécharger les nouvelles versions des programmes qu'il aurait trouvé, car c'est winget qui fait le travail, et la meuh automatise cette utilisation de winget et rends le processus très simple pour n'importe quel utilisateur.

Merci pour votre lecture et votre confiance !