

# Analyse de code et logiciels malveillants avec des outils de rétro-ingénierie logicielle (SRE)

## Table des matières

<b>IDA {Pro} – Tour du propriétaire .....</b>	<b>1</b>
Exo1 .....	3
Exo2 .....	3
Exo3 .....	3
Exo4 .....	3
Executables .....	4
<b>Ghidra .....</b>	<b>4</b>
Petit tour du propriétaire.....	4
Premiers Malwares.....	11
SYMBOL TREE .....	11
RECHERCHE DE CHAINE .....	11
OSINT .....	12
<b>Investigations libres (Bonus).....</b>	<b>14</b>

## NOTE :

Je vous recommande fortement d'utiliser une VM « vierge » (ou du moins PAS votre VM préférée ou machine principale) car on n'est jamais trop prudent. En effet, nous allons manipuler des fichiers malveillants, et, même si pour ce TP l'analyse sera statique, il existe des malwares qui utilisent les outils de SRE (Software Reverse Engineering) comme vecteur de déclenchement / propagation (ce qui est plutôt malin) :

[https://www.cvedetails.com/vulnerability-list/vendor\\_id-662/product\\_id-57094/NSA-Ghidra.html](https://www.cvedetails.com/vulnerability-list/vendor_id-662/product_id-57094/NSA-Ghidra.html)

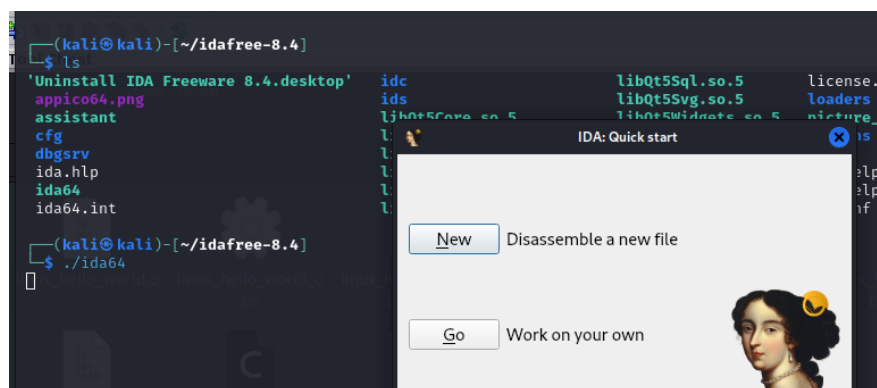
## IDA {Pro} – Tour du propriétaire

Un autre outil très célèbre est IDAPro, qui est décliné en plusieurs versions, dont une gratuite qui va nous intéresser aujourd'hui. Comme Ghidra que l'on verra plus tard, il offre des fonctionnalités de décompilation, de débogage, et supporte une vaste gamme d'architectures pour la reverse engineering. La version gratuite dispose de fonctionnalités limitées, mais elle est suffisante pour du hobbying et/ou pour prendre en main l'outil. A noter qu'il existe une

version intermédiaire (Home) qui est « accessible » (env. 300 € / an contre 1800 € min pour la version Pro).

IDA Free est disponible ici (vous devez vous inscrire) : <https://hex-rays.com/ida-free/#download>

Pour GNU/Linux, vous allez télécharger un .run. Pour rappel, lancer un .run, il faut souvent le rendre exécutable (chmod +x) et le lancer comme un script bash (./monfichier.run). Vous aurez alors une procédure d'installation qui vous demandera d'accepter des conditions d'utilisation et de choisir un emplacement d'installation (par défaut dans votre /home). Une fois la procédure terminée, vous devriez pouvoir lancer le programme en vous rendant dans le répertoire d'installation et en lançant l'appli :



L'interface d'IDA Pro ressemble un peu à celle de Ghidra, on retrouve des éléments en commun, comme, la fenêtre de fonctions :

Functions		
Function name	Segment	Start
__init_proc	.init	0000000000400480
sub_4004D0	.plt	00000000004004D0
puts	.plt	00000000004004E0
__stack_chk_fail	.plt	00000000004004F0
printf	.plt	0000000000400500
__libc_start_main	.plt	0000000000400510
strcmp	.plt	0000000000400520
__gmon_start__	.plt	0000000000400530
_start	.text	0000000000400540
deregister_tm_clones	.text	0000000000400570
register_tm_clones	.text	00000000004005A0
__do_global_ctors_aux	.text	00000000004005E0
frame_dummy	.text	0000000000400600
get_pwd	.text	0000000000400620
compare_pwd	.text	000000000040067A
main	.text	0000000000400716
__libc_csu_init	.text	0000000000400760
__libc_csu_fini	.text	00000000004007D0
_term_proc	.fini	00000000004007D4
puts	extern	0000000000601068
__stack_chk_fail	extern	0000000000601070
printf	extern	0000000000601078
__libc_start_main	extern	0000000000601080
strcmp	extern	0000000000601088
__gmon_start__	extern	0000000000601090

Idem pour les différentes représentations du code, à travers notamment sa version HEX (Onglet Hex View), une représentation du code en version graphique (IDA View) ou encore ses options, accessibles via le bandeau en haut de l'interface.

Différence par rapport à Ghidra, la présence d'une représentation graphique du code qu'on audit, bien utile pour mieux se repérer. Prenez quelques minutes pour investiguer les différentes couleurs (notamment les fonctions représentées en vert)



Votre objectif va être d'analyser des fichiers (pour le coup, des .exe) et de trouver des FLAGS. Pour rappel, un FLAG est une chaîne de caractères faisant preuve de la réussite d'un challenge en cybersécurité. Votre objectif va donc être de retrouver les flags des binaires. Vous allez voir que le format des flags est assez... explicite...

**NOTE : il est possible (et même recommandé pour les premiers niveaux), de ne pas utiliser QUE IDA. Vous pouvez en effet utiliser des commandes GNU/Linux, des programmes vus dans d'autres TP (notamment pour faire de l'analyse dynamique) ou même des outils/codes de votre choix. Pour chaque exercice, prenez des notes et détaillez bien votre méthode d'investigation.**

### Exo1

On commence facile... Essayez de lancer le programme dans un terminal pour voir ce qu'il fait...

### Exo2

Une fois le programme lancé, essayez de l'utiliser. Ensuite, posez-vous la question suivante : Comment fonctionne algorithmiquement le programme / Comment feriez-vous si vous deviez l'implémenter ? Une fois cette réflexion faite, essayez de trouver une commande, un outil ou une méthode permettant de récupérer l'information.

Comment feriez-vous pour rendre ce programme plus robuste (de manière spéculative, pas besoin d'implémenter vos réflexions) ?

### Exo3

Les outils de RSE sont vos amis. Essayez de récupérer **le mot de passe**. Note : Il est bien sûr possible de récupérer l'information avec d'autres outils ou commandes (notamment celle que vous auriez pu utiliser dans l'exercice précédent). De plus, si vous avez des soucis avec l'installation d'IDA et que vous voulez utiliser autre chose que Ghidra, vous pouvez utiliser Radare2 (commande r2 déjà présente dans Kali).

Comment feriez-vous pour rendre ce programme plus robuste (de manière spéculative, pas besoin d'implémenter vos réflexions) ?

### Exo4

Celui-ci est une variante de l'exo3, mais cette fois la chaîne à trouver (**qui est aussi un mot de passe**) est plus difficile d'accès (pensez à lancer le programme pour en apprendre un peu plus). Encore une fois, utilisez IDA Pro pour tenter de récupérer l'information. Note : Là encore, on peut utiliser d'autres outils pour aller plus vite, mais comme notre objectif est de mieux comprendre les RSE, autant prendre son temps avec.

## Executables

Procédez à l'analyse des exécutables « Executable1 » et « Executable2 » pour tenter de récupérer les flags.

## Ghidra

Ghidra <https://ghidra-sre.org/> est développé par la NSA (National Security Agency). C'est un logiciel open source compatible avec beaucoup d'architectures de processeurs / OS destiné à l'analyse de programmes {malveillants}.

Pour l'installer sur GNU/Linux (saveur Debian), rien de plus simple (attention : cela peut être un peu long en fonction des serveurs utilisés...) :

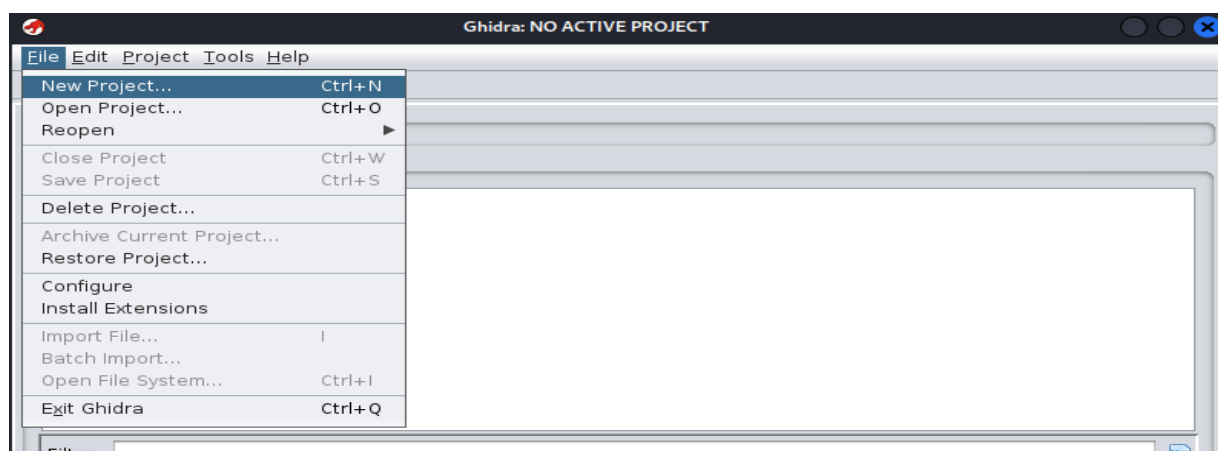
```
(kali㉿kali)-[~]  
$ sudo apt install ghidra  
[sudo] password for kali:  
Reading package lists ... Done  
Building dependency tree ... Done
```

Pour les autres OS, vous pouvez jeter un œil à la documentation qui est bien développée : [https://htmlpreview.github.io/?https://github.com/NationalSecurityAgency/ghidra/blob/Ghidra\\_11.0.1\\_build/GhidraDocs/InstallationGuide.html](https://htmlpreview.github.io/?https://github.com/NationalSecurityAgency/ghidra/blob/Ghidra_11.0.1_build/GhidraDocs/InstallationGuide.html)

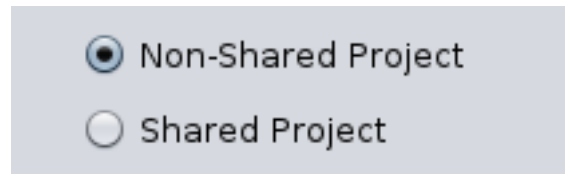
Une fois installée et lancée, l'application peut lever une exception « `IllegalArgumentException` ». C'est ce qu'il s'est passé dans mon cas (Kali 6.5.0 vanilla), sans impact pour l'utilisation...

## Petit tour du propriétaire...

Ghidra est organisé sous forme de projets (à l'extension .gpr). Commençons par créer un nouveau projet :



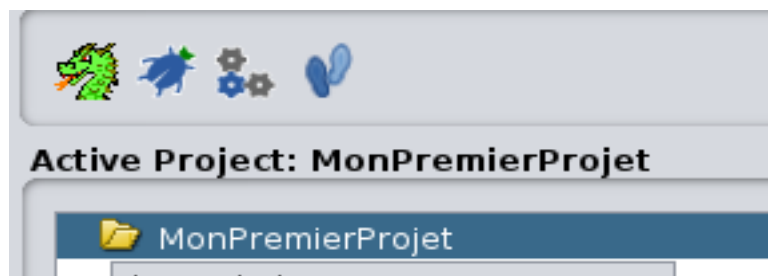
Pas la peine de faire du partagé dans notre cas, un **non-shared project** suffit.



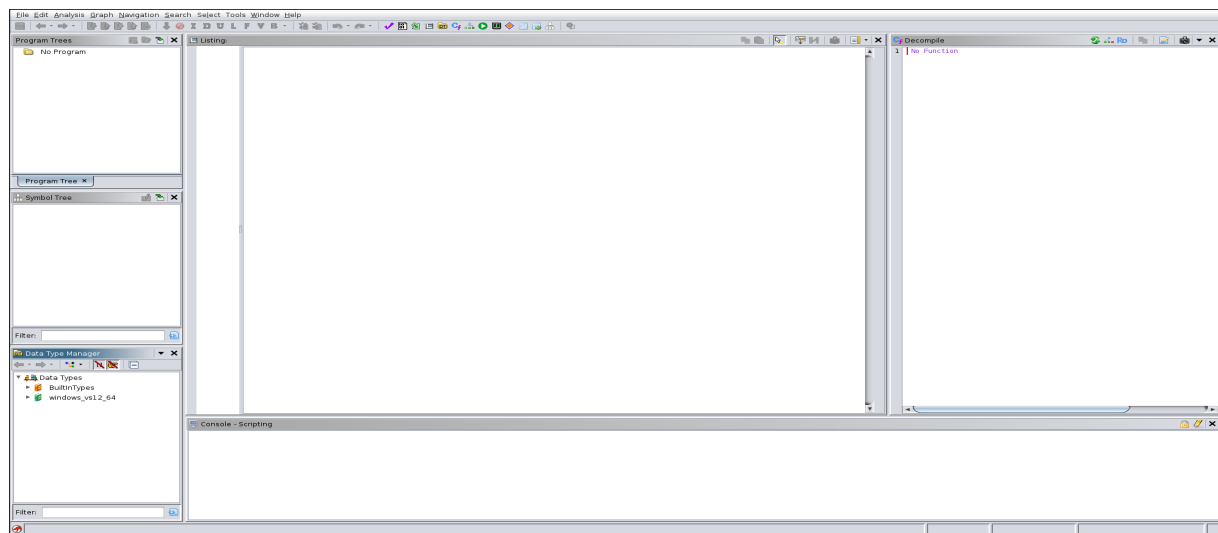
Une fois les autres informations usuelles saisies, on peut créer le projet.



Une fois le projet actif créé, il est possible d'interagir avec lui de plusieurs manières via les 4 icônes :



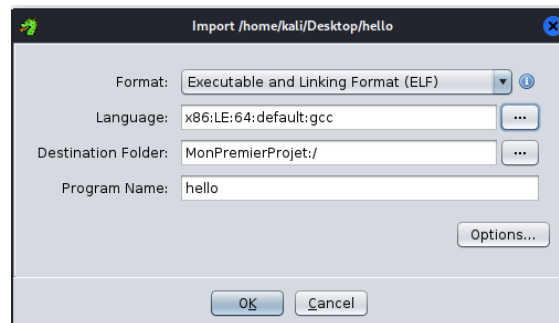
Nous allons cliquer sur le dragon vert pour lancer l'explorateur de code. Après une petite animation, on arrive sur l'interface suivante :



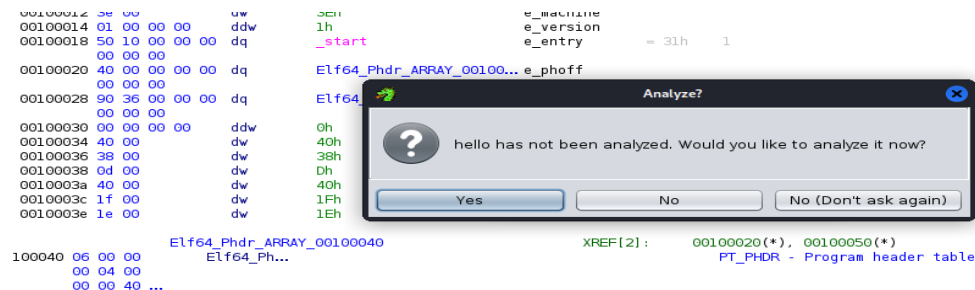
Notre projet est pour l'instant vide, nous allons lui donner du code à manger. Pour cela, il faut appuyer sur la touche « i » (on peut aussi aller sur « File » > « Import File »).

Pour le code, nous allons utiliser un traditionnel HelloWorld.c que nous allons d'abord **compiler**. Si vous n'avez plus le fichier, prenez quelques minutes pour le refaire. Une fois importé, Ghidra vous donne des options. En général, il est préférable de les laisser par défaut,

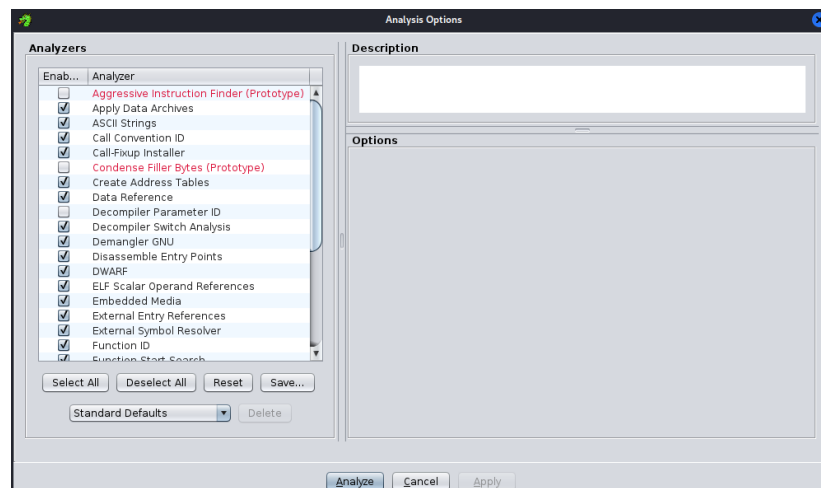
mais il est possible de modifier à la main le langage et le format si besoin (par exemple, s'il se plante et vous propose une mauvaise extension ou un mauvais langage) :



Une fois le bouton OK pressé, Ghidra nous demande si on veut analyser notre exécutable. Là encore, il est toujours intéressant d'accepter la proposition :

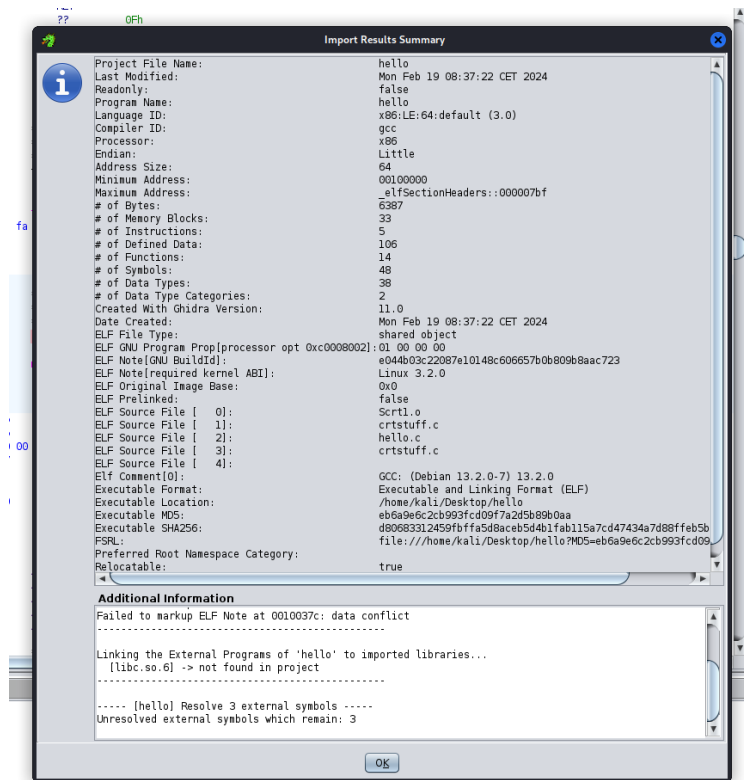


Ghidra nous propose des options d'analyse, les options en rouge sont des options en beta (prototypes) qui peuvent être instables (ou produire des résultats incohérents). Là encore, nous allons laisser les options par défaut :

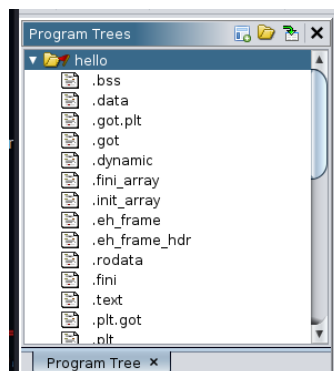


Une fois l'analyse réalisée (en principe très rapide avec notre petit programme helloworld), Ghidra nous donne un résumé qui permet d'avoir des informations intéressantes, comme :

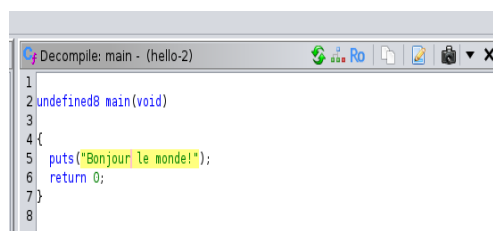
- Des métriques (taille, nombre d'instructions, nombre de symboles, etc.)
- Les hashes (MD5 et SHA256)
- Des méta-informations (date, fichiers créés, type de CPU, compilateur, etc.)



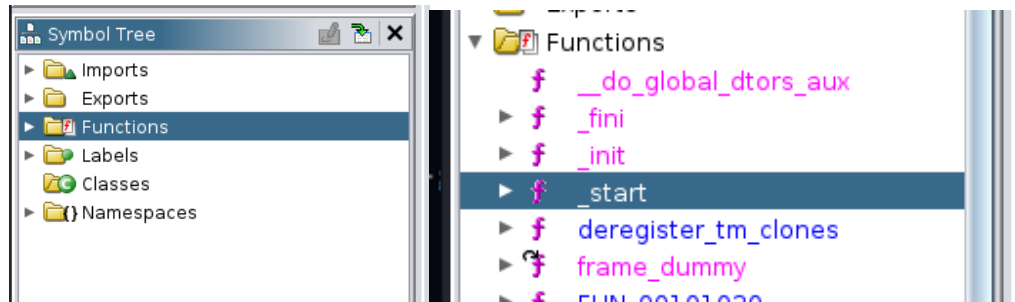
Une fois la popup récapitulative fermée, vous pouvez voir que des informations sont apparues dans notre projet. Par exemple, vous pouvez voir en haut à gauche (**Program Tree**), l'arborescence des éléments d'assembleur (e.g. .data, .text, .bss, etc.). En double cliquant sur un élément, vous devriez avoir le détail qui s'affiche à droite, dans la fenêtre principale :



La partie droite (**Decompile**) permet d'avoir une vue du code décompilé. Cette vue est intéressante car elle permet de voir le programme sous un aspect plus haut niveau :

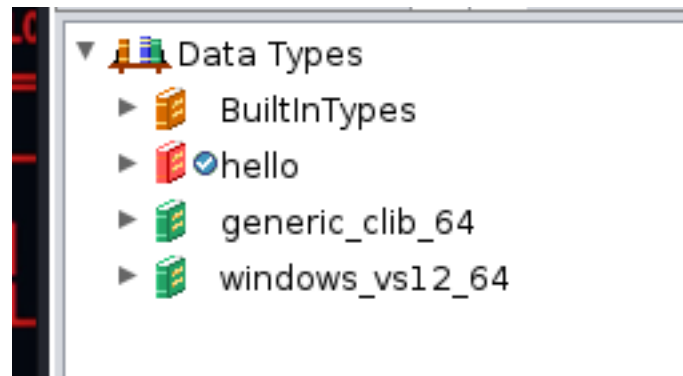


Une autre vue intéressante est la vue **Symbol Tree**, qui permet d'organiser notre programme par éléments sémantiques, comme les fonctions ou les classes (au sens Programmation Orientée Objets). On retrouve ici aussi des concepts évoqués précédemment, comme le fameux **\_start** qui permet, une fois que l'on clique dessus, de s'orienter automatiquement dans le code :

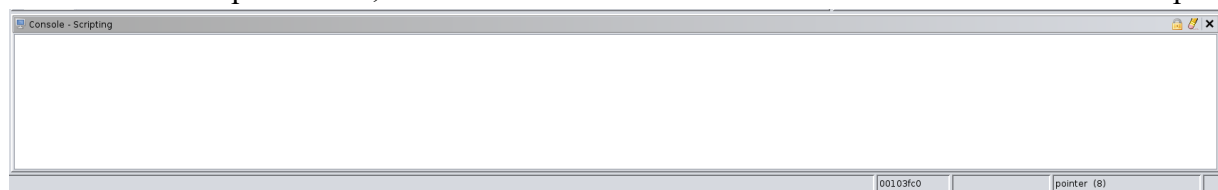


En dessous de cette partie, on retrouve enfin la **vue par Données**, qui permet, comme son nom l'indique de définir, gérer et appliquer des structures de données plus ou moins complexes à notre projet. En général, la structure est organisée de la manière suivante :

- En **orange (builtin)** on retrouve tous les types de données usuelles (char, int, byte, etc.)
- En **rouge**, les types de données qui ont été relevés par l'analyse de notre projet
- En **vert**, les autres types de données chargés par Ghidra (par exemple des archives/librairies de Visual Studio 12 comme on peut le voir sur la capture d'écran suivante).

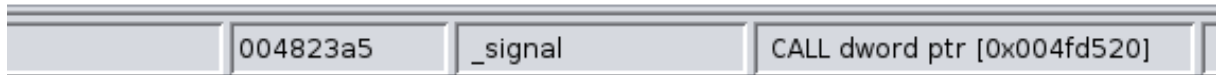


Au bas de l'application vous pouvez enfin trouver la console, qui permet d'afficher des informations pertinentes, notamment lors de l'exécution de scripts.

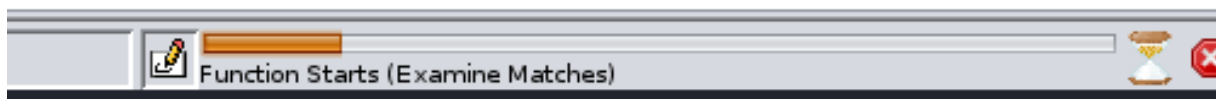




Cette partie de l'écran permet aussi de se repérer en donnant l'adresse mémoire (0048...) la fonction (\_signal) et l'instruction (CALL dword ptr [0x004fd520]) sur lesquelles notre curseur pointe.



De plus, la console permet aussi de voir le traitement en cours, utile pour savoir si notre analyse n'a pas planté...



Enfin, la barre d'icône permet d'effectuer plusieurs opérations de recherche, navigation, manipulation, analyse et affichage.

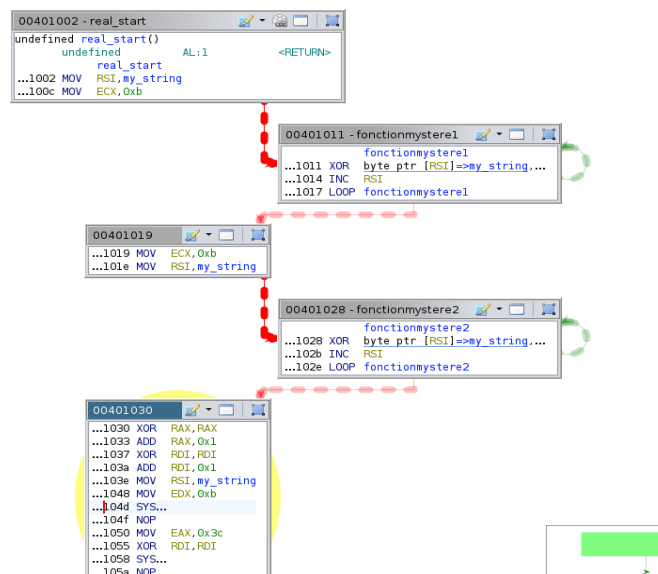


Par exemple, il peut être pertinent, lors de l'analyse d'un programme, de visuellement représenter son exécution. Pour cela, on peut utiliser l'icône graphe de fonction (en général il

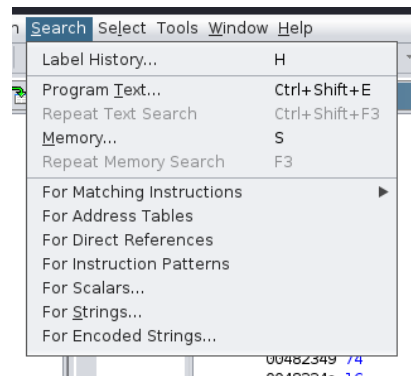
faut mettre en surbrillance le code en le surlignant pour que le graphe apparaisse)



Voici par exemple un programme mystère, dans lequel vous pouvez voir les fonctions mystères constituées de boucles :

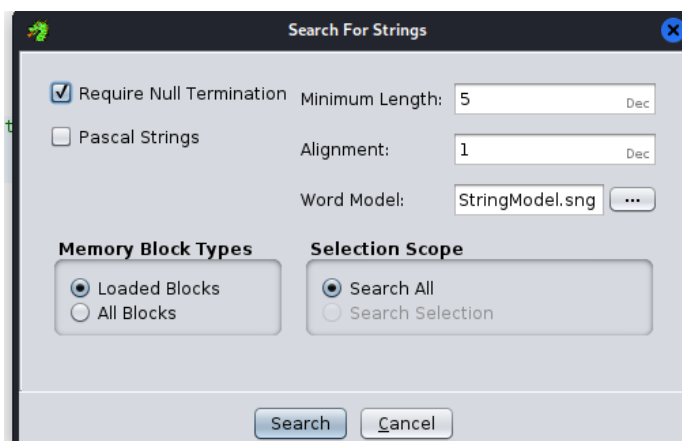


Autre fonctionnalité intéressante de Ghidra : la recherche ! Il est en effet possible d'effectuer plusieurs types de recherche via le menu idoine en haut :



1. **Address Table Search:** Permet de rechercher des tables d'adresses, pour trouver des pointeurs ou des adresses dans le code.
2. **Direct Reference Search:** Offre la possibilité de trouver toutes les références directes à un objet spécifique, tel qu'une fonction ou une variable, permettant de voir où et comment un élément est utilisé dans le code.
3. **Instruction Patterns Search:** Aide à localiser des séquences spécifiques d'instructions dans le code, utile pour identifier des morceaux de code qui exécutent des opérations particulières ou des constructions algorithmiques connues.
4. **Scalar Search:** Permet de rechercher des valeurs scalaires spécifiques dans le code, telles que des nombres « hardcodés ».
5. **Strings Search:** Facilite la recherche de chaînes de caractères littérales (URL, nom commun, etc.)

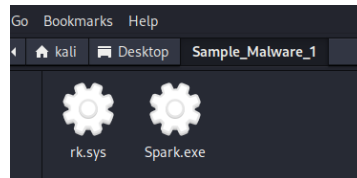
Pour la plupart de ces fonctionnalités, il est possible d'affiner ses recherches. Par exemple, dans le cas de la recherche par Strings, on peut sélectionner des modèles de mots. Ghidra propose un modèle par défaut (StringModel.sng) qu'il est possible d'enrichir / changer en fonction de ses besoins :



## Premiers Malwares

Maintenant que nous avons pris en main les principales interfaces de Ghidra, nous allons ouvrir un fichier malveillant.

Notre petit malware est disponible dans une archive que je vous vais vous communiquer durant le cours (n'hésitez-pas à me demander si besoin). Il est composé de 2 fichiers :



L'objectif va être d'utiliser Ghidra pour en apprendre plus sur ces fichiers...

Pour cela, vous allez commencer par faire un nouveau projet (vous pouvez l'appeler comme vous voulez, comme premier\_malware par exemple) et importer le **.exe** à l'intérieur.

Utilisez l'analyse automatique de Ghidra (vous pouvez laisser les options par défaut). Le programme étant plus complexe que notre Hello World (et peut être plus complexe que les applications que vous avez testé par vous-même), le temps d'analyse peut-être un peu long, mais c'est un « mal » nécessaire pour en gagner plus tard (rappelez-vous par ailleurs que vous avez la barre de progression en bas à droite pour vous notifier qu'il y a du traitement en cours).

## SYMBOL TREE

Une fois l'analyse terminée, une bonne approche est de commencer par regarder le **Symbol Tree**. Comme nous l'avons évoqué, ce dernier permet de nous donner des infos concernant les fonctions utilisées, et notamment les fonctions importées, qui permettent souvent de glaner des indices sur la capacité du programme / ce qu'il sait faire (lire des buffers, enregistrer des fichiers, créer des sockets, utiliser des protocoles Web, lancer des services, etc.)

Jetez un œil aux différentes DLL utilisées et, à partir des noms de méthodes, essayez de voir les capacités du programme en justifiant vos hypothèses.

## RECHERCHE DE CHAÎNE

Utilisez la fonction **Search for Strings** pour rechercher différentes informations, comme :

- les programmes qui pourraient interagir avec notre malware
- des noms d'utilisateurs ou de machines
- des informations intéressantes
- etc.

Là encore, prenez des notes et essayez de justifier vos hypothèses.

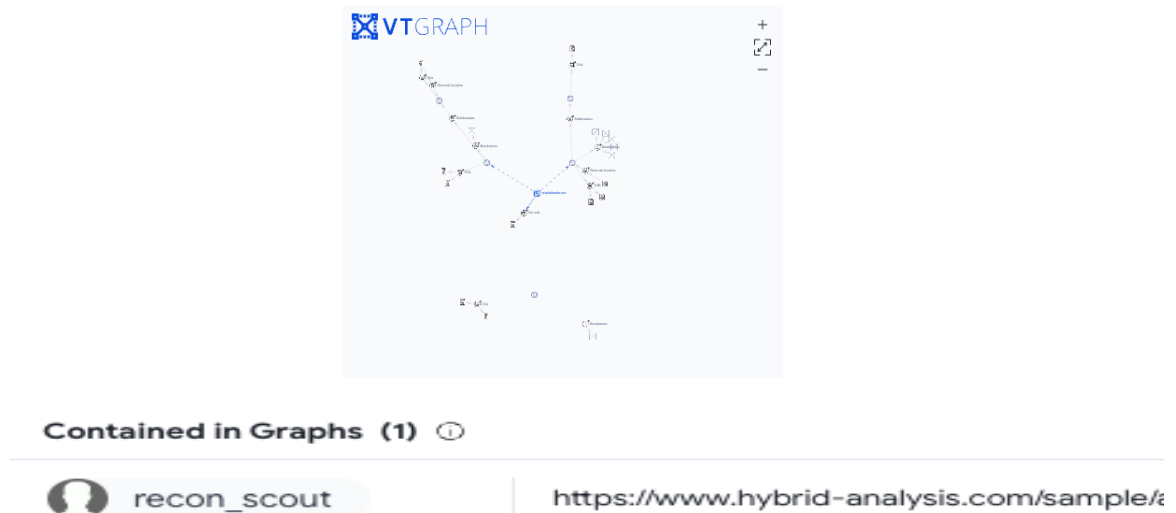
## OSINT

L'OSINT (OpenSource Intelligence) est une technique permettant de glaner des informations sur une cible (une IP, une personne, une machine, etc.) à partir de sources en accès libre (Internet, base de données publiques, méta informations). L'OSINT est une technique souvent utilisée en amont d'un test d'intrusion ou d'une véritable attaque qui a l'intérêt d'être indirecte : c'est-à-dire que l'on n'interagit pas directement avec la cible. La reverse engineering peut utiliser de l'OSINT pour corroborer une hypothèse.

Essayez par exemple de rechercher des extensions Web usuelles (.fr, .net etc.) dans les **chaines**. Si vous trouvez des choses intéressantes, rendez-vous sur Virustotal, onglet URL pour glaner plus d'infos !

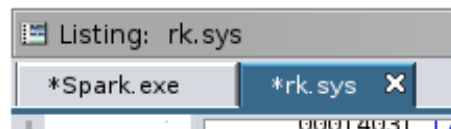


Notamment, vous devriez pouvoir trouver dans les **commentaires**, un lien vers **hybrid-analysis** et un graph VirusTotal, qui pourrait vous permettre de récupérer encore plus d'informations...



Pensez encore une fois à prendre des notes pour étayer votre rapport.

Regardons maintenant le deuxième fichier (rk.sys). Pour cela, il faut comme pour le fichier précédent, l'importer dans votre projet et laisser Ghidra l'analyser. Vous devriez ainsi avoir vos 2 fichiers :

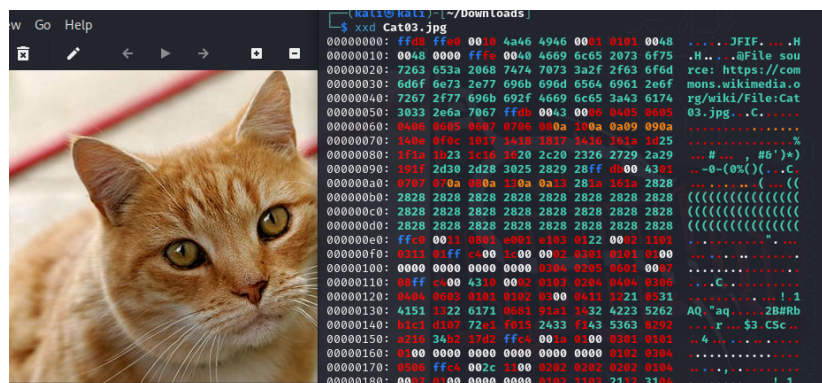


Lancez la fenêtre permettant de visualiser le fichier sous forme hexa :



En bas de cette fenêtre, vous devriez voir les adresses de départ et de fin de ce fichier. Quelles sont ces adresses ?

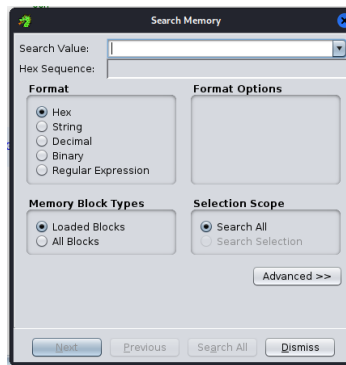
On peut se repérer dans un fichier hexa à l'aide des entêtes et des en-queues, que l'on appelle des magic numbers. En effet, les fichiers aux extensions usuelles (ex : jpg, word, etc.) disposent d'une signature bien définie. Par exemple, en ouvrant une image avec xxd (un lecteur/éditeur Hexa en CLI), on peut voir que le fichier commence par ffd8, qui est le nombre magique du format jpg. Ainsi, si cette image est contenue dans un fichier plus grand, je pourrai retrouver sa trace via cet en-tête, même si l'image est « noyée » dans le reste de l'information.



On peut facilement retrouver les nombres magiques des formats les plus connus, notamment ici : [https://en.wikipedia.org/wiki/List\\_of\\_file\\_signatures](https://en.wikipedia.org/wiki/List_of_file_signatures)

Ces informations acquises, essayez maintenant de chercher à quoi correspond l'entête du fichier rk.sys.

Une fois cela fait, regarder dans le fichier spark.exe s'il y a des occurrences de ce magic number. Pour cela, vous pouvez utiliser l'outil « **Search Memory** » et cliquer sur le bouton « **Search all** ». Notez les infos si jamais vous trouvez des occurrences et essayer de comprendre à quoi correspondent ces occurrences...



Comme pour l'exercice précédent, vous pouvez vous amuser avec d'autres fonctionnalités de Ghidra pour en apprendre plus sur le programme.

## Investigations libres (Bonus)

S'il vous reste du temps, vous pouvez vous rendre sur les liens suivants pour télécharger des Malwares de votre choix et investiguer (attention, ce sont de vrais malwares, donc à ne pas utiliser sur votre machine principale) :

- **MalwareBazaar** : <https://bazaar.abuse.ch/browse/>
- **TheZoo** : <https://github.com/ytisf/theZoo>
- **VX-Underground**: <https://vx-underground.org/Samples>

Vous pouvez aussi tester d'autres SRE comme :

- **Radare2**: <https://github.com/radareorg/radare2>
- **Cutter**: <https://cutter.re/>
- **Any Run** (online platform): <https://app.any.run/>
- **Relyze** : <https://www.relyze.com/index.html>