Initiation à la programmation bas niveau sous Windows

Cet atelier a pour but d'exercer ses participants à la conception de programmes en assembleur sous Windows.

Objectifs pédagogiques

- Savoir utiliser le jeu d'instructions des processeurs modernes, tels que l'Intel IA-32.
- Prendre connaissance de quelques-uns des mécanismes internes de Windows.
- Comprendre les interactions entre DLL de sous-systèmes et le noyau de Windows (éventuellement par le biais des appels systèmes).

Thèmes abordés

- **Langage d'assemblage** Langage de bas niveau directement compréhensible par le processeur.
- **API Windows** Ensemble normalisé de fonctions permettant de se servir des fonctionnalités des différents systèmes d'exploitation de la gamme Windows.
- **Portable Executable** Format de fichier sous-jacent aux exécutables et bibliothèques sur les systèmes d'exploitation Windows 32 et 64 bits.
- **Architecture de Windows** Sous-systèmes d'environnement, DLL de sous-système (Kernel32.dll, Advapi.dll, etc.), bibliothèque générale de support système (Ntdll.dll), services natifs, services de l'exécutif, noyau (Ntoskrnl.exe), et bien d'autres.

En dehors des éléments techniques, cet atelier est fait pour que vous rencontriez concepts et terminologie. Identifiez les termes que vous ne comprenez pas et entreprenez des recherches à leur sujet.

Outils

- **Masm** Assembleur Microsoft épaulant divers projets de la firme, dont la suite de développement Visual Studio. Consultez le site http://www.ollydbg.de pour plus de détails et http://www.ollydbg.de/odbg110.zip pour le téléchargement.
- **OllyDbg** Outil d'analyse de binaires au niveau utilisateur. Consultez le site http://masm32.com pour plus de détails et http://website.assemblercode.com/masm32/masm32v11r.zip pour le téléchargement.

Travaux pratiques : injection de code

L'injection de code désigne une méthode par laquelle un programme se rend maitre des instructions exécutées dans le contexte d'un autre programme. Une telle technique peut s'établir de deux façons, et être de nature statique ou dynamique. Une injection est qualifiée de statique lorsqu'elle a lieu avant l'exécution d'un programme (a fortiori avant la transformation de celui-ci en processus). A l'inverse, une injection de type dynamique se distingue par le fait de cibler un processus en cours d'exécution.

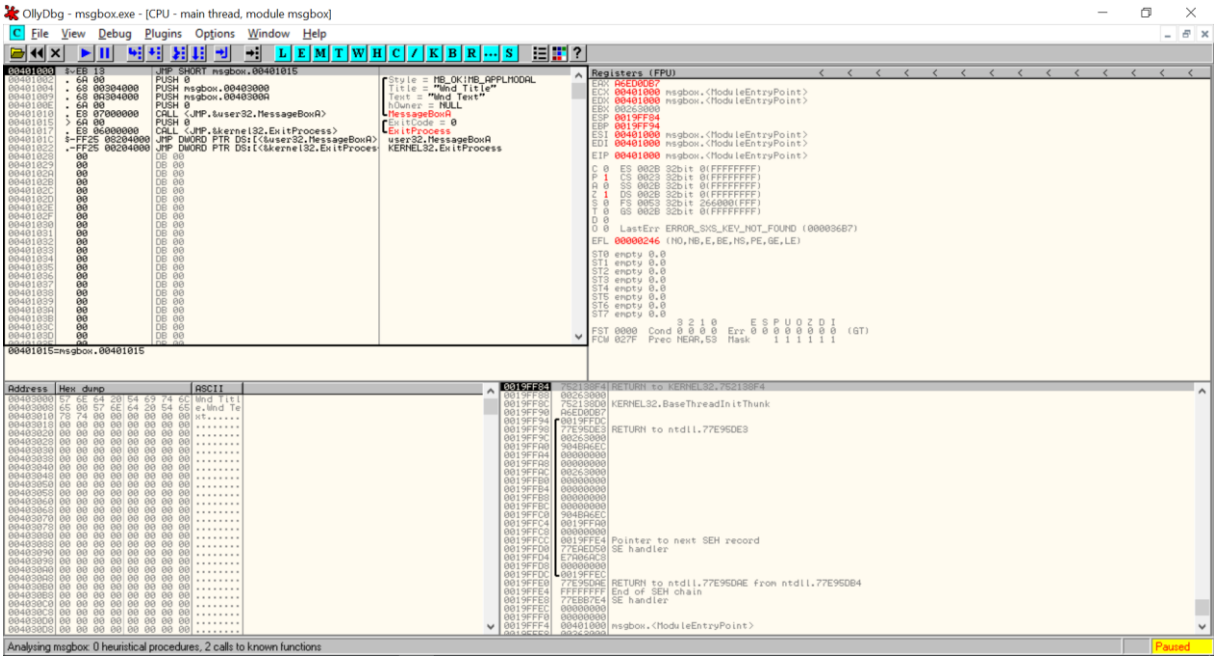
Nous allons dans ce qui suit explorer l'injection de code statique. A cet effet, nous manipulerons le programme msgbox de sorte que son exécution intègre une boite de dialogue affichant un message.

Afin de manipuler le programme cible, nous utiliserons l'outil d'analyse automatique nommé OllyDbg, que vous pouvez obtenir depuis l'URL suivante : http://www.ollydbg.de.

Décompressez odbg110.zip dans un dossier de votre choix.

A partir du chemin d'installation adéquat, démarrez OllyDbg (OllyDbg.exe). Lors du premier lancement, OllyDbg présente éventuellement à l'utilisateur un message demandant la mise à jour de certaines bibliothèques (.dll). Cliquez sur Non.

Allez dans File, Open puis sélectionnez le fichier msgbox1.exe. Dans le menu Debug, cliquez sur Run. Vous devriez à ce moment voir le code machine exécutable dudit programme.



- **E1** Prenez note de la valeur du registre EIP, à quoi correspond dans le cas présent le point d'entrée de l'exécutable (0x00401000 dans notre exemple).
- **E2** Prenez note de la première instruction immédiatement après le point d'entrée du module. (0x00401002 dans notre exemple).

Du fait de l'alignement, l'espace d'adressage virtuel afférent à tout processus contient un certain nombre de zones mémoire n'hébergeant aucune information utile. Dans les grandes lignes, l'injection statique consiste justement à profiter de la présence de ces régions afin d'y insérer diverses instructions.

Au niveau de la fenêtre des instructions de OllyDbg (juste en dessous du menu), faites défiler vers le bas jusqu'à repérer une zone remplie de zéros.

- **E3** Prenez note de l'adresse mémoire du premier octet à constituer cette zone une zone remplie de zéros. (0x00401028 dans notre exemple).

Nous allons maintenant ajouter un appel à la fonction MessageBox de Windows, lequel se présente sous la forme suivante :

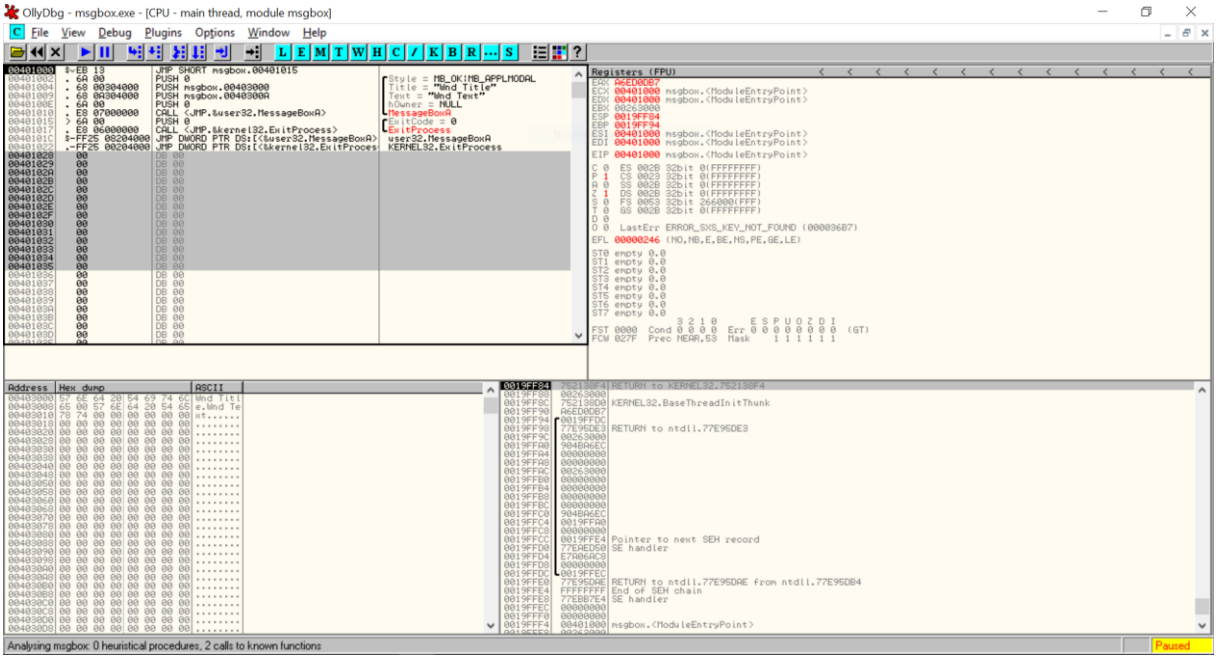
```
int WINAPI MessageBox(

  _In_opt_ HWND    hWnd,

  _In_opt_ LPCTSTR lpText,

  _In_opt_ LPCTSTR lpCaption,

  _In_     UINT    uType
```

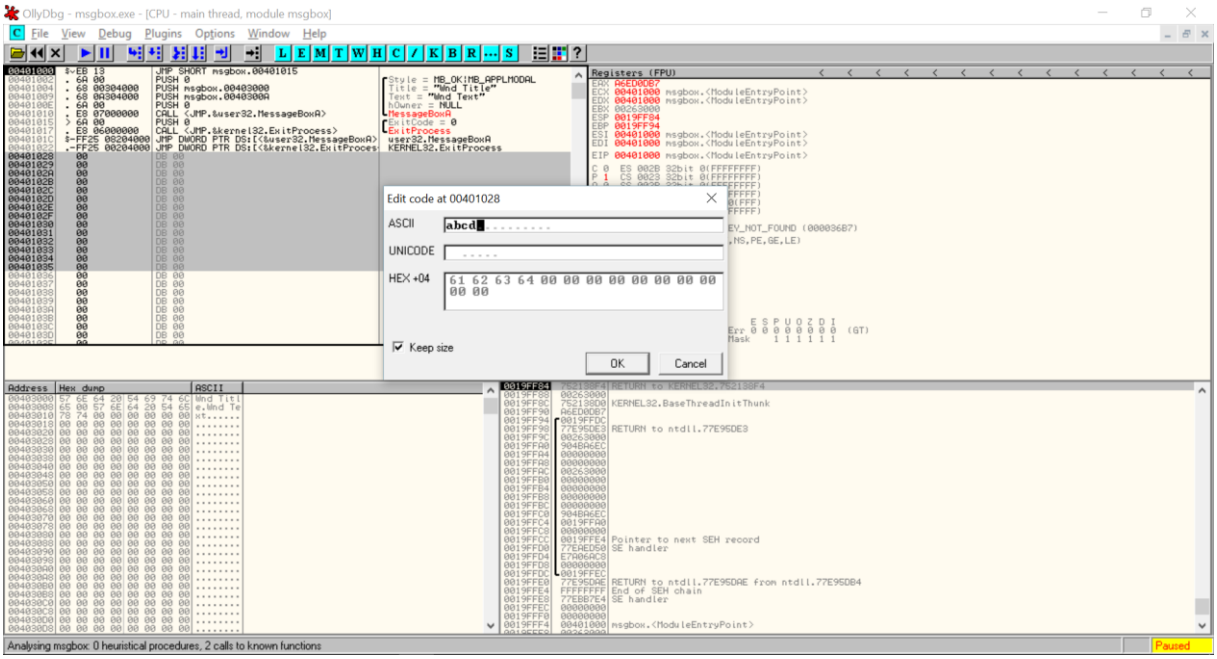
```
);
```

Le pseudo code en assembleur correspondant à cet appel est :

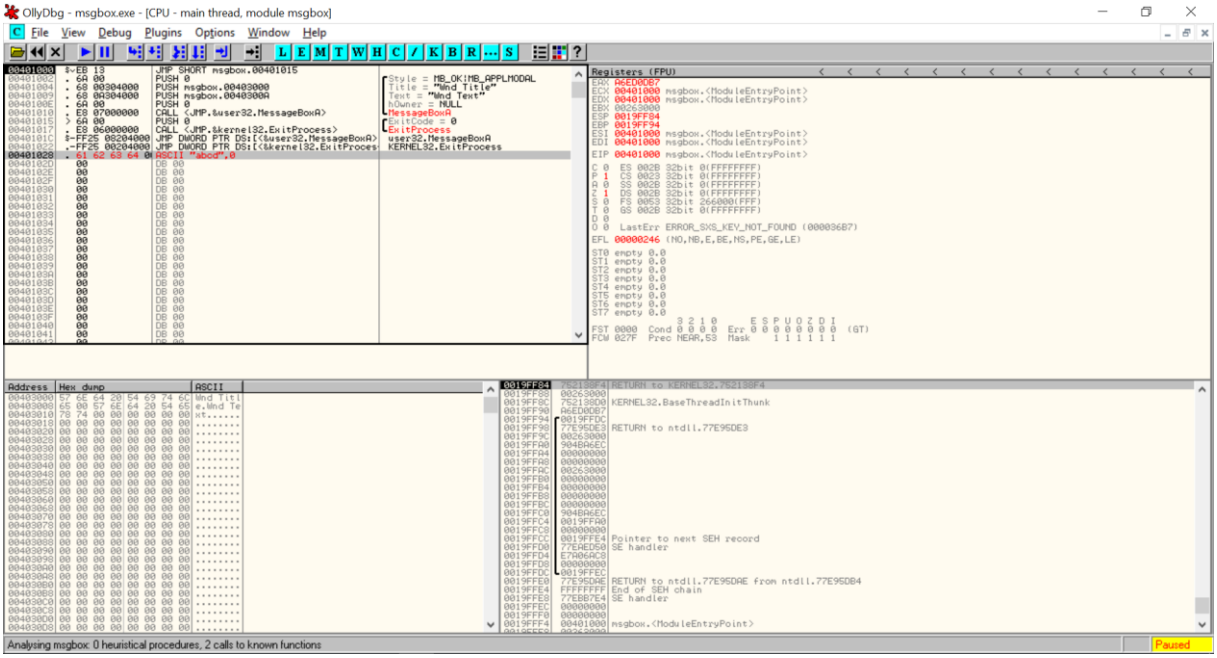
```
push 0

push "Titre de la fenetre"

push "Texte de la fenetre"

push 0

call User32.MessageBoxA
```

Nous allons maintenant ajouter le code via OllyDbg. Mettez en surbrillance un groupe d'octets parmi la zone. Faites un clic droit et sélectionnez Binary->Edit. Dans le champ ASCII de la fenêtre nouvellement ouverte, inscrivez une chaine de caractères quelconque.





Faites CTRL+A de sorte à permettre à OllyDbg d'analyser une nouvelle fois l'exécutable.



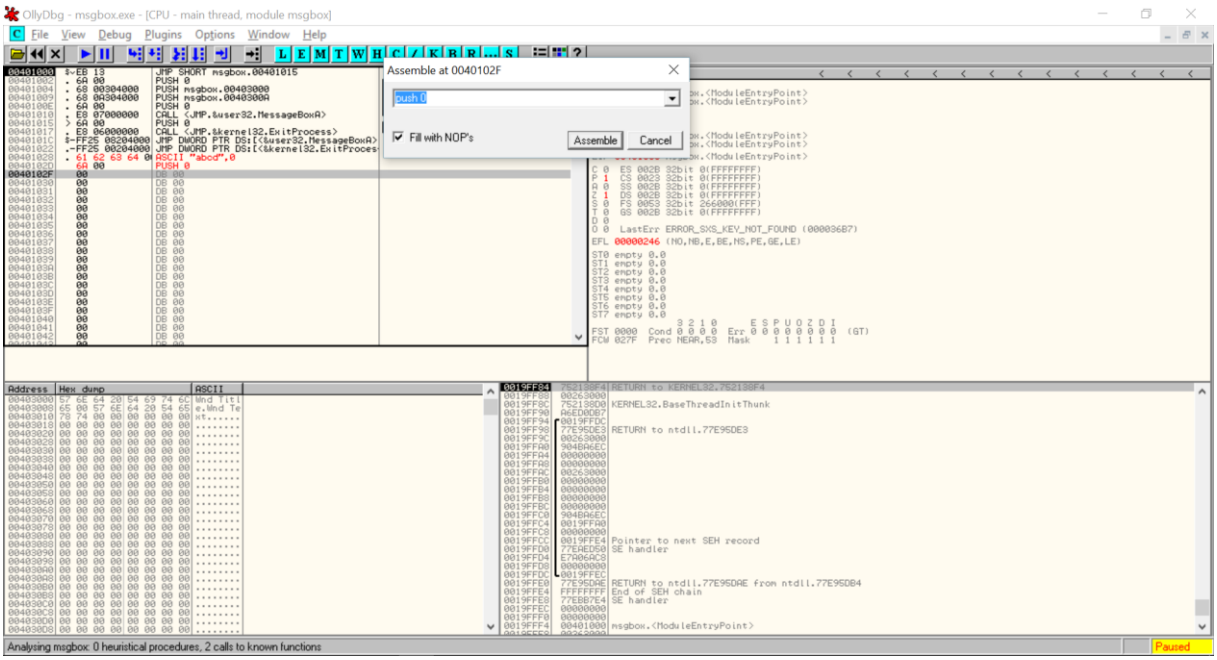
E3 Prenez note de l'adresse mémoire de cette chaine de caractères.

- **E4** Prenez note de l'adresse mémoire du premier octet à constituer cette chaine de caractères. (0x00401028 dans notre exemple).

Double-cliquez sur un des champs DB 00 en dessous de la chaine que vous venez d'ajouter. Vous devriez voir apparaitre une fenêtre ouvrant la voie à l'assemblage. Dans celle-ci, inscrivez :

```
push 0
```

Cliquez sur le bouton Assemble.

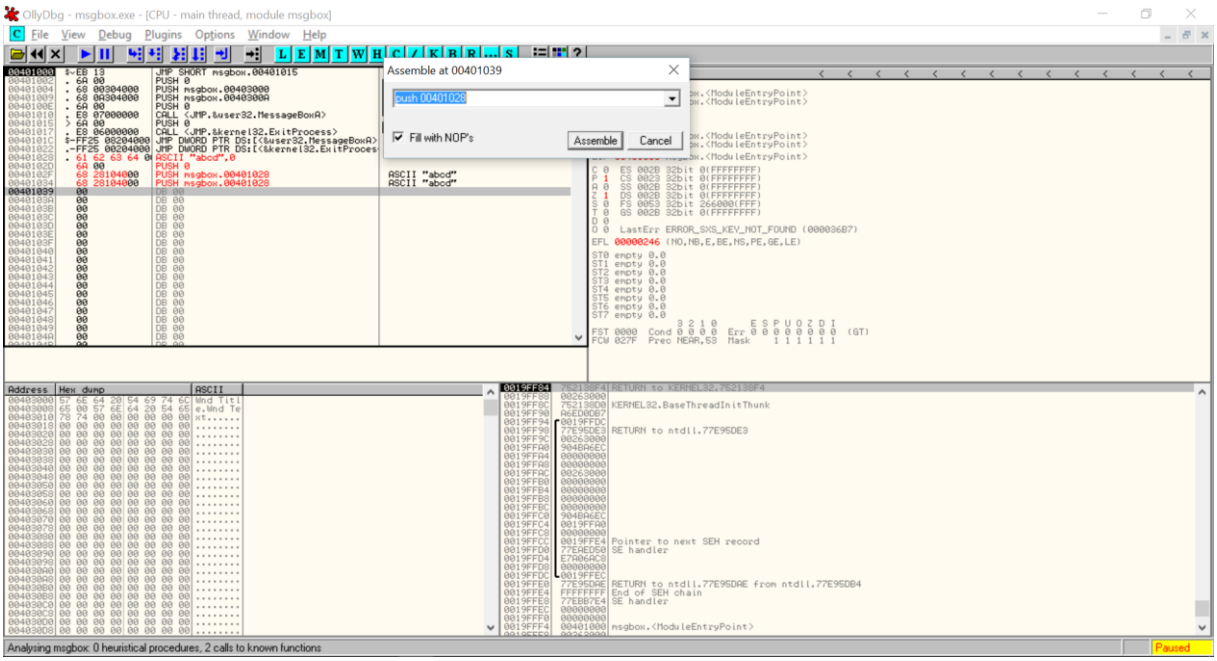


- **E5** Prenez note de l'adresse mémoire du premier octet à constituer ces instructions. (0x0040102d dans notre exemple).

Inscrivez ensuite :

push adresse

où *adresse* est l'adresse mémoire de la chaine de caractères précédemment constituée (voir E4). Cliquez deux fois sur le bouton Assemble, de sorte à former dans un même mouvement les paramètres lpText et lpCaption pour MessageBox.

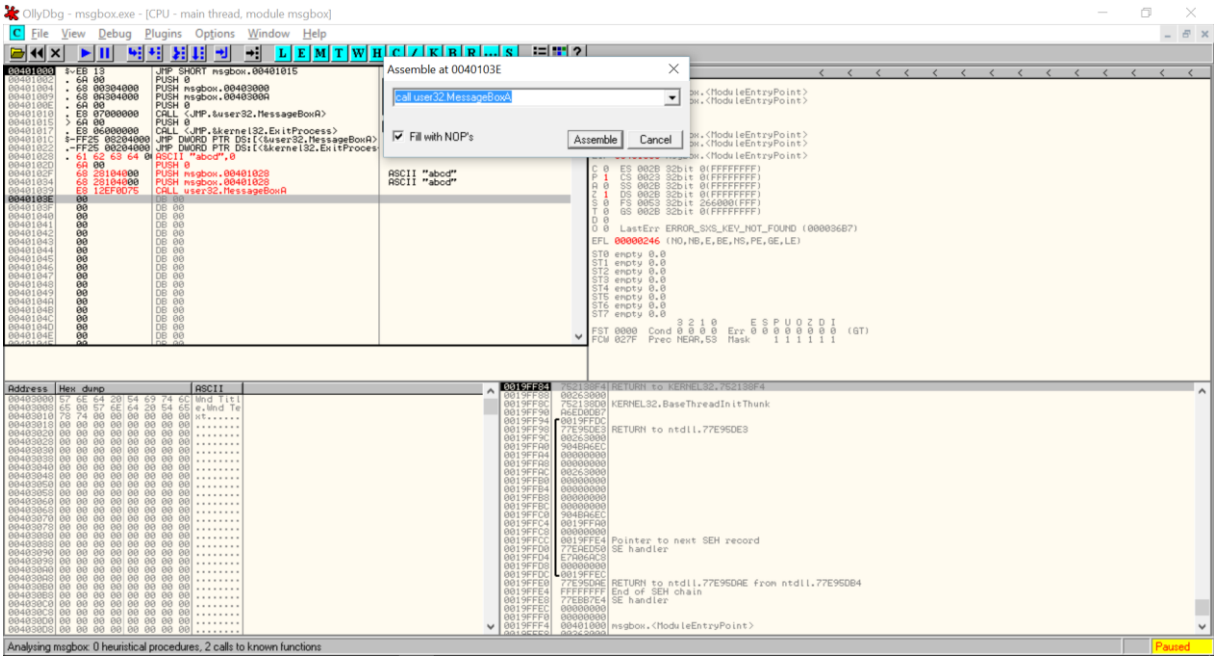


Inscrivez ensuite le quatrième et dernier argument de la fonction.

push 0

Ecrivez ensuite ce qui suite puis sur le bouton Assemble :

call user32.MessageBoxA



Une fois ceci fait, inscrivez :

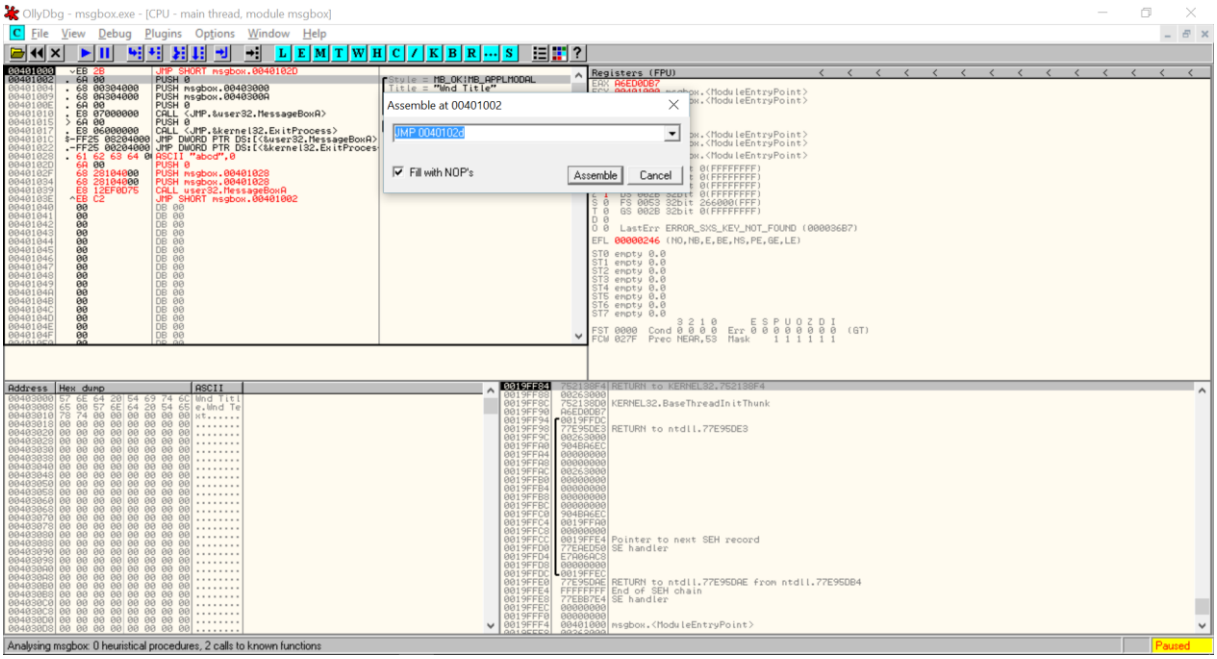
jmp adresse

où adresse est l'adresse de la seconde instruction après le point d'entrée du module (voir E2).

Fermez la fenêtre Assemble. Revenez à l'origine du programme (E1) et double cliquez sur l'instruction de saut qui s'y trouve. La fenêtre Assemble apparait de nouveau. Inscrivez

jmp adresse

où adresse est l'adresse mémoire du premier octet à constituer vos instructions (voir E5).



Revenez à l'origine du programme et faites Run. Deux boites de dialogue devraient se succéder, la première résultant des instructions ajoutées, la seconde de l'exécution « normale » du processus.

Exercices

Le but des exercices qui suivent est d'écrire en assembleur un programme mettant en œuvre les différentes procédures vues à la section Travaux pratiques.

- **ex1** Injecte une boite de dialogue dans le programme msgbox1.exe, lequel contient par ailleurs un appel à user32!MessageBoxA et une instruction de saut, les deux étant là afin de vous faciliter la tâche.
- **ex2** Injecte une boite de dialogue dans le programme msgbox2.exe, qui contient une instruction de saut mais pas de références à la fonction Windows MessageBox.
- **ex3** Injecte une boite de dialogue dans le programme msgbox3.exe, qui ne contient ni instruction de saut ni références à la fonction Windows MessageBox. Conseil : regardez à quoi correspond le champ de l'entête PE nommé AdressOfEntryPoint.
- **ex4** Injecte une boite de dialogue dans un programme Windows quelconque, et cela indépendamment de la version du système d'exploitation utilisée.

Consignes de rendu

Les réponses aux différents exercices présentés en amont doivent être envoyées par mail à l'adresse maillard.arnaud@gmail.com. Pas de consignes précises à ce niveau. Si vous utilisez des directives d'assemblage particulières, précisez-le.

Consignes diverses

- Toutes les facilités d'utilisation introduites par MASM (par exemple .IF, .WHILE, etc.) sont interdites.
- En ce qui concerne les opérations sur les fichiers, vous devez utiliser le mécanisme des fichiers représentés en mémoire (MapViewOfFile et consorts).
- Tout appel de fonction externe hors du sous-système Windows est interdit. Utilisez par exemple HeapAlloc plutôt que malloc.