OllyDbg

Tutoriel sur le débuggeur gratuit OllyDbg
Par Dark Jedi (http://www.securityhack.net), enrichi par Jérôme Athias

1° : Le téléchargement

Cliquez ici.
C'est le site officiel, cliquez sur Download et choisissez votre version tout en bas
Il est indiqué que c'est un shareware, mais vous n'êtes pas obligé(e) de le payer ni quoi que se soit, il fonctionne parfaitement sans être bridé pour autant.

2° : L'installation

Vous allez obtenir une archive zip du style odbg<version>.zip
Dézippez-la
Ouvrez le dossier créé et lancez OLLYDBG.EXE
Nous allons le configurer.

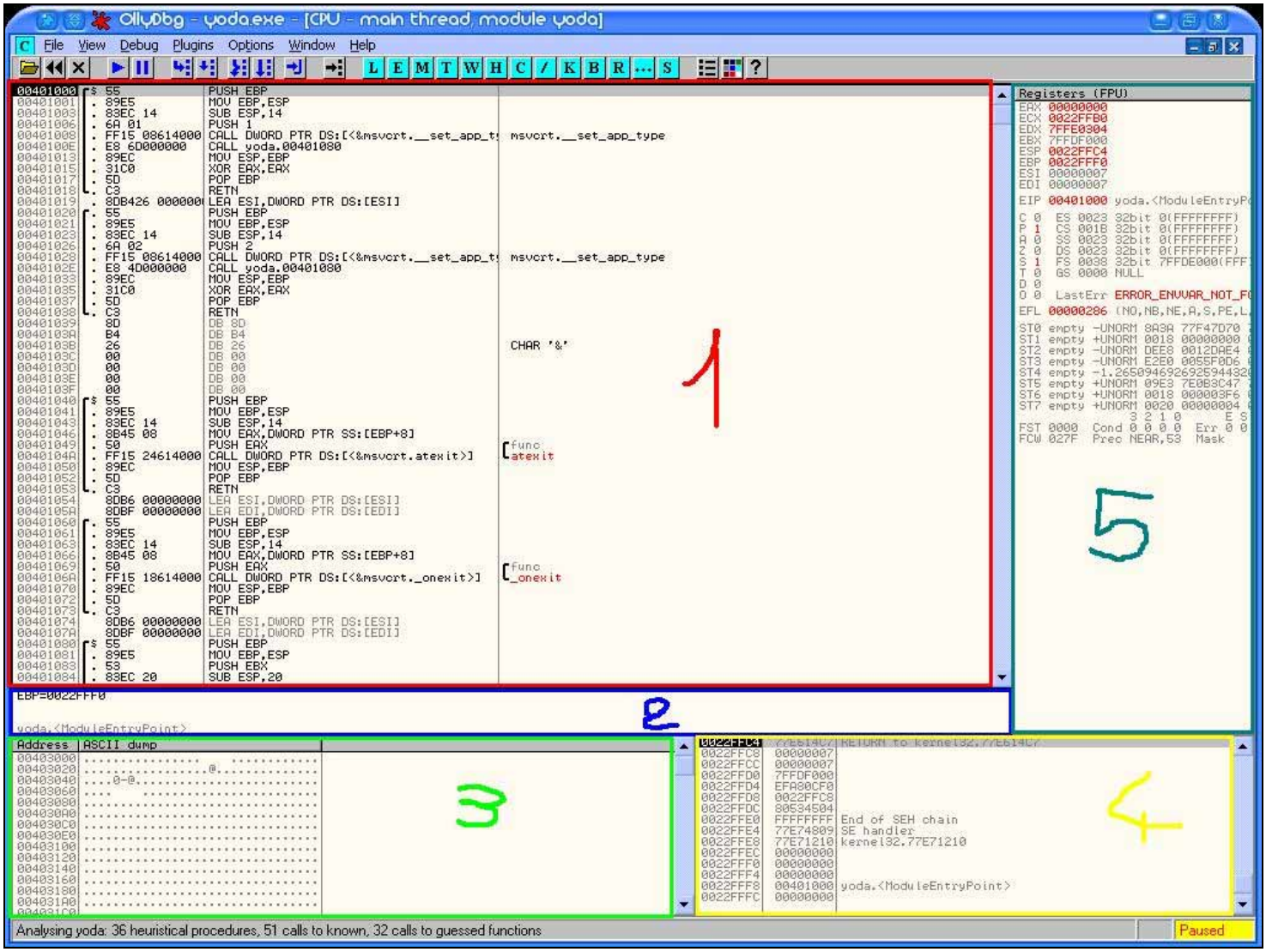
3° : Configuration

OllyDbg est ouvert

Cliquez sur le menu "Options" puis sur "Add to Explorer", puis ensuite sur "Add OllyDbg to menu in Windows Explorer", et finalement sur "Done"
Dorénavant quand vous ferez un clic droit sur un fichier exécutable (*.exe), vous aurez dans le menu une option : "Open with OllyDbg", en cliquant dessus vous ouvrez OllyDbg.

4° : Présentation de l'interface

Le fichier désassemblé que vous verrez sur les prochains screenshots est mon crack me (appelé Yoda.exe) que je crackerai avec vous dans mon prochain article.
Téléchargez-le ici.
J'ouvre le fichier comme indiqué au dessus.



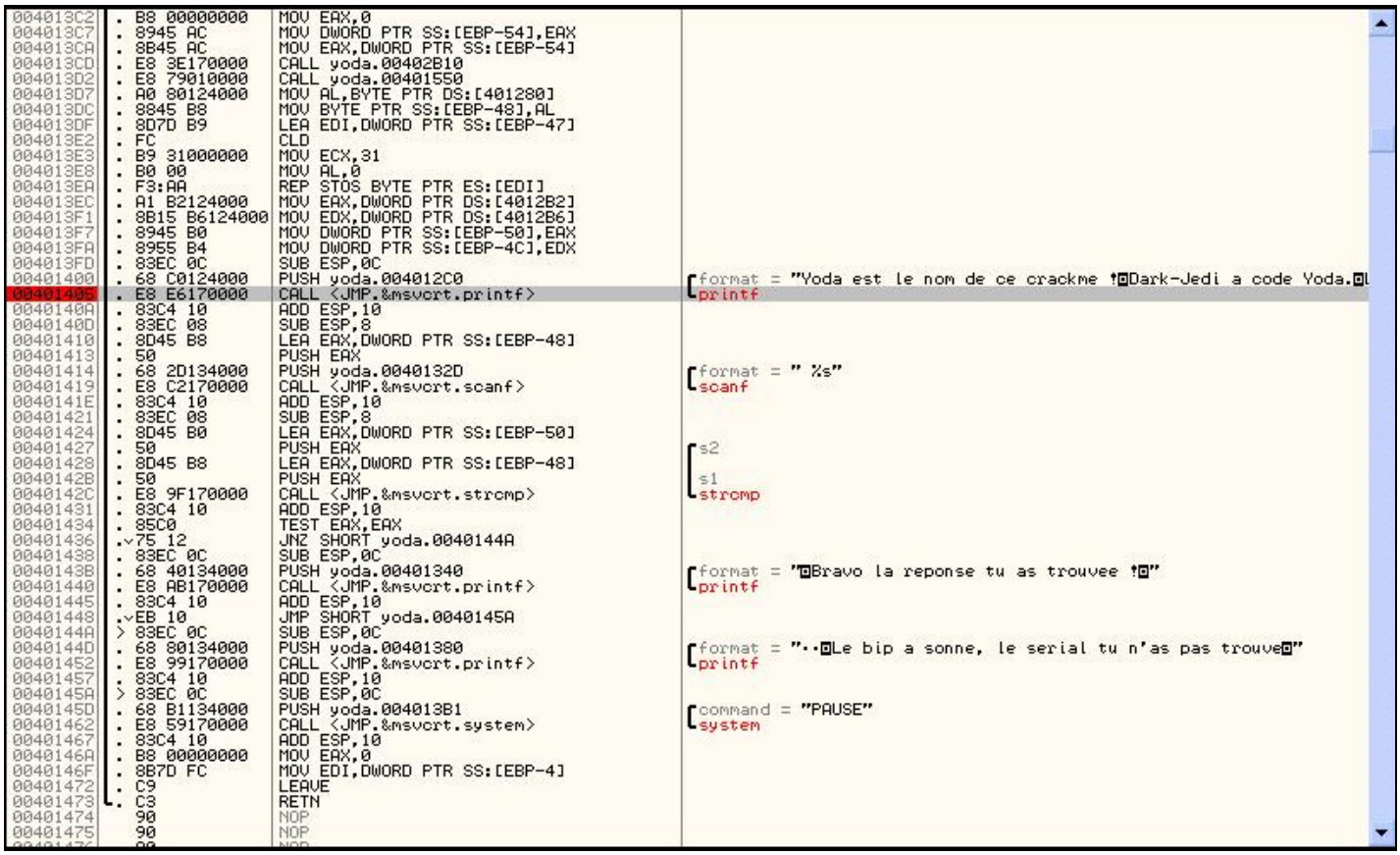
Et là vous voyez pleins de trucs bizarres.
(J'ai entouré les zones en couleur et attribué un numéro à chacune pour mieux les voir)

Zone 1 : Code asm
Zone 2 : Rappel de l'état des opérandes (paramètres qui sont utilisés par une instruction), très pratique !
Zone 3 : Mémoire, affiche la mémoire utilisée par le programme, peut être affichée sous différentes formes (ici en ASCII)
Zone 4 : La pile
Zone 5 : De haut en bas, les registres, les flags, la pile pour le processeur arithmétique.

Maintenant regardez en haut de la zone 1 (code asm), vous voyez
00401000 | 55 | PUSH EBP
Nous avons l'offset (00401000), ensuite l'équivalent en tokens de PUSH EBP qui est l'instruction.
Vous pouvez remarquez que l'offset sur cette instruction a un fond noir, contrairement aux autres, sa veut dire que c'est la prochaine instruction qui va être exécutée. Nous y reviendrons plus tard quand je vous aurais expliqué les breakpoints.

5° : Les breakpoints

Ceux qui connaissent un peu le C savent que printf() sert à afficher quelque chose et que scanf() sert à récupérer ce qui est entré au clavier.
Nous allons descendre un peu dans le listing asm et arriver sur cette partie :



Vous voyez une suite de printf() et un scanf() et ensuite un system() et la ligne dessus chaque instruction, OllyDbg nous affiche les paramètres comme par exemple pour system(), la vraie fonction est system(pause).
J'ai placé un breakpoint (littéralement, un point de pause) sur l'offset 00401405 (premier printf), vous pouvez le remarquer par l'apparition d'un fond rouge, pour en placer un c'est simple : cliquez sur l'instruction, appuyez sur F2 et voilà ;)
Pour l'enlever, même manipulation, on peut en placer plusieurs dans un programme.
Mais ne l'enlevez pas pour le moment :p
Je vais maintenant lancer l'exécution normale du programme en appuyant sur F9.
Il fait une pause comme je l'avais indiqué, le fond noir indique que cette instruction sera la prochaine à être exécutée.

6° : Traçage pas à pas

Il y a aussi une invite de commande qui c'est ouvert, elle est vide pour l'instant.
Nous allons la remplir avec le premier printf(), retournez sous OllyDbg et faites F8.
Le pointeur noir a avancé d'une instruction seulement grâce à l'usage de F8 et le printf() c'est exécuté et seulement celui-ci.
Nous pourrions continuer à tracer avec F8 pour arriver au scanf() mais nous allons poser un bp (nom court de BreakPoint) juste après scanf() et enlever celui sur le premier printf().
Et maintenant faites F9, le programme "break".
Tapez un serial et faites ENTREE.

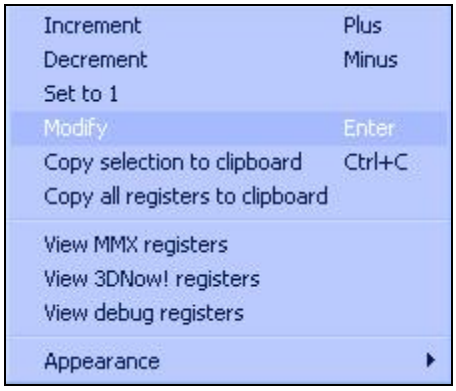
Revenez dans OllyDbg, nous pouvons maintenant examiner, entre autre, la mémoire et les registres juste après la récupération du serial.
Au lieu de faire F8, on peut aussi utiliser F7, la différence est que F7 va entrer dans les call, je m'explique : (pseudo-code)

```
01 | CALL 05
02 | MOV EAX, ESI
03 | CMP EAX, ECX
04 | JMP SHORT 08
05 | ADD ESP,10
06 | INC ESI
07 | RETN
08 | ...
```

Si vous faites F8 sur l'offset 01 vous allez passer directement à l'offset 02 (mais le call sera fait de manière "invisible")
Si vous faites F7 vous allez atterrir à l'offset 05 comme indiqué par le call et arrivé au retn, vous allez revenir sur l'offset 02, et vous aurez pu voir en détail ce que fait cette routine. (Évitez les F7 sur les appels aux dll système comme scanf, vous n'allez sûrement pas comprendre grand chose et vous n'avez pas besoin de savoir comment fonctionne scanf pour comprendre le programme !)

7° : Mémoire, registres, pile

Vous pouvez modifier la valeur des registres comme par exemple :



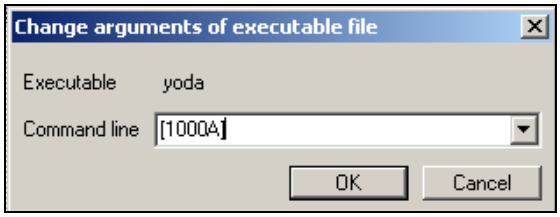
En faisant un clic droit sur un registre et en choisissant Modify ou en faisant directement un double-clic.
Vous pouvez faire de même dans les zones 2 et 4.
Vous pouvez afficher la pile selon EBP et ESP, faites un clic droit dessus et "Address->Relative to EBP OU ESP"

8° : Divers

OllyDbg est un programme très complet, vous pouvez faire beaucoup d'autres choses avec.
Je vais même vous montrer dans un prochain article comment cracker "Yoda" (mon premier crack me), le plus proche possible (pour ne pas donner la réponse quand même lol !) du premier niveau du BigContest avec Olly ;)

9° : Passage d'arguments

Il sera parfois utile d'appeler le programme que vous debuggez en lui passant des paramètres.
Cliquez sur le menu « Debug » puis sur « Arguments »



Vous pouvez ici saisir les arguments en les séparant par un espace.
Dans les cas de l'étude de débordements (type Buffer Overflows), il est intéressant de pouvoir passer en paramètre une grande quantité de caractères, ceci peut se faire comme suit :
[quantitécaractère]
Ainsi sur l'exemple si dessus on passera 1000fois le caractère « A » en argument (ce qui n'est pas utile pour yoda.exe ;)