

OllyDbg

Tutoriel sur le débogueur 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"

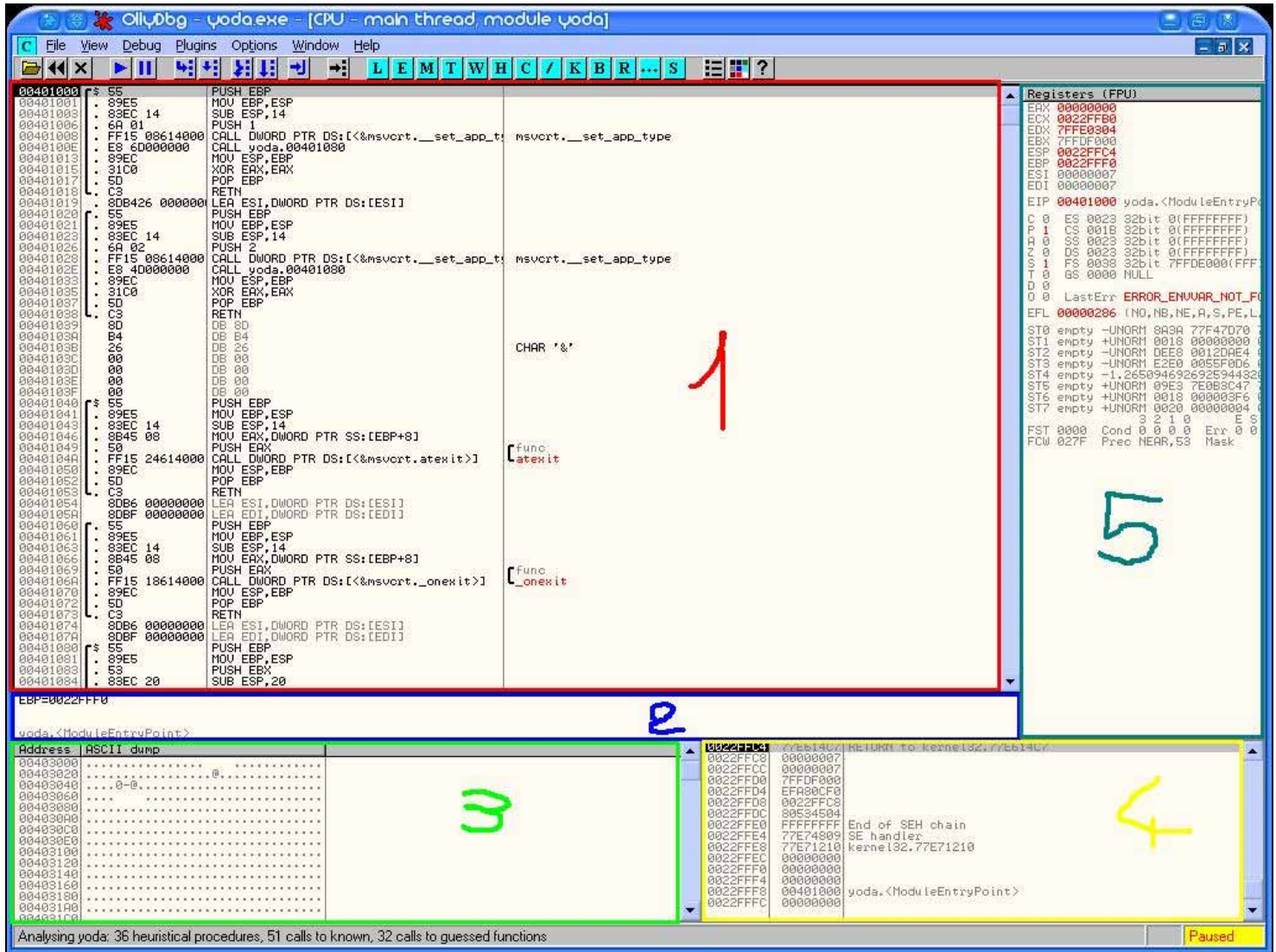
Dorénavant quand vous ferez un clic droit sur un fichier exécutable (*.exe), vous aurez dans le menu une option : "Open with OllyDbg", en cliquant dessus vous ouvrez OllyDbg.

4° : Présentation de l'interface

Le fichier désassemblé que vous verrez sur les prochains screenshots est mon crack me (appelé Yoda.exe) que je crackerais 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 opérands (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 remarquer 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 :

```
004013C2 . B8 00000000 MOV EAX,0
004013C7 . 8945 AC MOV DWORD PTR SS:[EBP-54],EAX
004013CA . 8B45 AC MOV EAX,DWORD PTR SS:[EBP-54]
004013CD . E8 3E170000 CALL yoda.00402B10
004013D2 . E8 79010000 CALL yoda.00401550
004013D7 . A0 80124000 MOV AL,BYTE PTR DS:[401280]
004013DC . 8845 B8 MOV BYTE PTR SS:[EBP-48],AL
004013DF . 8D7D B9 LEA EDI,DWORD PTR SS:[EBP-47]
004013E2 . FC CLD
004013E3 . B9 31000000 MOV ECX,31
004013E8 . B0 00 MOV AL,0
004013EB . F3:AA REP STOS BYTE PTR ES:[EDI]
004013ED . A1 B2124000 MOV EAX,DWORD PTR DS:[4012B2]
004013F1 . 8B15 B6124000 MOV EDX,DWORD PTR DS:[4012B6]
004013F7 . 8945 B0 MOV DWORD PTR SS:[EBP-50],EAX
004013FA . 8955 B4 MOV DWORD PTR SS:[EBP-4C],EDX
004013FD . 83EC 0C SUB ESP,0C
00401400 . 68 C0124000 PUSH yoda.004012C0
00401405 . E8 E6170000 CALL <JMP.&msvcrt.printf> [format = "Yoda est le nom de ce crackme ! Dark-Jedi a code Yoda."]
0040140A . 83C4 10 ADD ESP,10 [scanf]
0040140D . 83EC 08 SUB ESP,8
00401410 . 8D45 B8 LEA EAX,DWORD PTR SS:[EBP-48]
00401413 . 50 PUSH EAX
00401414 . 68 2D134000 PUSH yoda.0040132D
00401419 . E8 C2170000 CALL <JMP.&msvcrt.scanf> [format = " %s"
0040141E . 83C4 10 ADD ESP,10
00401421 . 83EC 08 SUB ESP,8
00401424 . 8D45 B0 LEA EAX,DWORD PTR SS:[EBP-50]
00401427 . 50 PUSH EAX
00401428 . 8D45 B8 LEA EAX,DWORD PTR SS:[EBP-48]
0040142B . 50 PUSH EAX
0040142C . E8 9F170000 CALL <JMP.&msvcrt.stromp> [s2
00401431 . 83C4 10 ADD ESP,10 [s1
00401434 . 85C0 TEST EAX,EAX [stromp]
00401436 . 75 12 JNZ SHORT yoda.0040144A
00401438 . 83EC 0C SUB ESP,0C
0040143B . 68 40134000 PUSH yoda.00401340
0040143E . E8 AB170000 CALL <JMP.&msvcrt.printf> [format = " Bravo la reponse tu as trouvee "
00401441 . 83C4 10 ADD ESP,10 [printf]
00401444 . EB 10 JMP SHORT yoda.0040145A
00401447 . 83EC 0C SUB ESP,0C
0040144A . 68 80134000 PUSH yoda.00401380
0040144D . E8 99170000 CALL <JMP.&msvcrt.printf> [format = ". Le bip a sonne, le serial tu n'as pas trouve"
00401452 . 83C4 10 ADD ESP,10 [printf]
00401455 . 83EC 0C SUB ESP,0C
00401458 . 68 B1134000 PUSH yoda.004013B1
0040145B . E8 59170000 CALL <JMP.&msvcrt.system> [command = "PAUSE"
0040145E . 83C4 10 ADD ESP,10 [system]
00401461 . B8 00000000 MOV EAX,0
00401464 . 8B7D FC MOV EDI,DWORD PTR SS:[EBP-4]
00401467 . C9 LEAVE
00401470 . C3 RETN
00401473 . 90 NOP
00401474 . 90 NOP
00401475 . 90 NOP
```

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

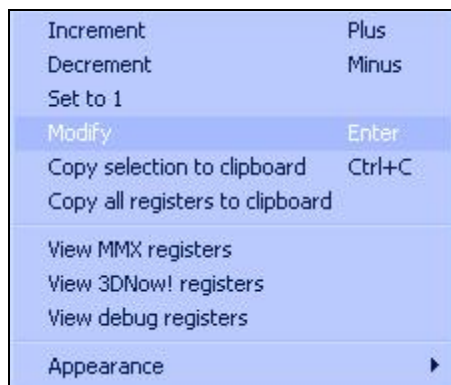
08 | ...

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"

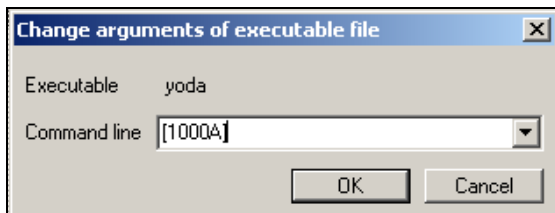
8° : Divers

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 ;)