

Olllydbg调试经验总结

一、一切的开始

1. 可以打开EXE或者DLL文件
2. 在打开文件的窗口那里，可以设置参数 Arguments
3. 可以使用 File->Attach 调试已经在运行的进程
4. 在 Options->Debugging options->Events 中设置进程加载后暂停的位置，默认是 winMain，可以设置成 System breakpoint，在所有代码开始执行前暂停。

二、不同的断点

1. 软件断点

一般调试器设置的都是这种断点，其实就是把设置断点位置的指令的第一个字节修改为 0xCC (INT 3)。

2. 硬件断点

由硬件DR寄存器判断是否到达断点位置，DR0 ~ DR3 寄存器存储断点地址，DR7 存储控制信息。这些寄存器内容容易被程序修改，DR7 中的 General Detect Flag 用于检测通过 mov 指令访问这些寄存器的情况。

- on execution: 执行到某句指令时暂停运行

3. 内存断点



分为硬件和软件两种断点形式

- on access: 访问到某个地址时暂停运行
- on write: 修改某个地址时暂停运行

4. 条件断点

是软断点，在条件满足时暂停运行

三、快捷键

1. 汇编代码处按 space 可以进行修改
2. 内存区域处按 ctrl+g 可以跳转到任意地址
3. 选定指令后，按 F2 可以添加/删除断点
4.  用来添加注释
5.  用来添加标签，类似IDA中的重命名，为一个地址添加标签，所有对该地址的引用都会变成这个标签

四、其他功能

1. Debug->Execute till user code 可以在迷失于库代码时使用。
2. 在调试恶意软件时，可以直接在 Debugging Options 中忽略所有 exception，因为有时恶意软件作者会为了增加调试难度故意添加一些异常。
3. 修改了部分指令之后，在代码窗口右键 Copy to executable->All modifications 可以对修改后的文件进行保存。
4. 在各子窗口地址的位置右键，Follow in Dump，可以在内存区域直接查看该地址处的内容

5. 在代码区域右键，`New origin here`，可以直接将程序的流程转向这里，这在分析shellcode时很有用
6. Memory Map功能(工具栏上的 `M` 按钮)，可以显示该进程中各导入文件，各段在内存中的分布情况。如果有rebase的情况发生，可以由此确定新的基地址，方便在IDA中同步。
7. Call Stack功能(工具栏上的 `K` 按钮)，可以查看函数调用情况
8. Trace into/over功能，和Step into/over功能类似，但是会对过程中各寄存器、标志的变化情况做记录。*我用的不多，这里只是为了防止忘记*
9. 有极少情况，使用 `step over` 的调试方式，如果之后的函数不执行 `ret` 或者直接把返回地址弹出栈了，调试就无法正常进行。
10. 如果想利用python脚本协助调试，需要使用ImmDbg软件

五、插件

1. OllyDump：在脱壳的时候很有用
2. Hide Debugger：专门用于处理反调试的一个插件
3. Command line：执行Ollydbg命令

命令	功能
BP <i>expression</i> [<i>condition</i>]	设置软断点。可以直接在某函数开始执行处设置 <code>bp gethostbyname</code>
BC <i>expression</i>	删除断点
HW <i>expression</i>	设置on execution硬断点
BPX <i>label</i>	在每次调用 <i>label</i> 时设置断点
STOP or PAUSE	暂停执行
RUN	执行
G [<i>expression</i>]	执行到某地址处
S	step into
S0	step over
D <i>expression</i>	dump某地址

4. Bookmarks: 在代码窗口右键，`Bookmarks->Insert Bookmark`，可以在该代码处添加书签，方便之后直接访问。

六、经验与教训

5. 分析某个函数的功能，可以在调用前后分别设置断点，然后查看执行前后寄存器、内存中的变化情况。
6. Ollydbg对于函数和参数的识别没有IDA那么强大，所以应该结合从两者得到的信息，协助对方的分析