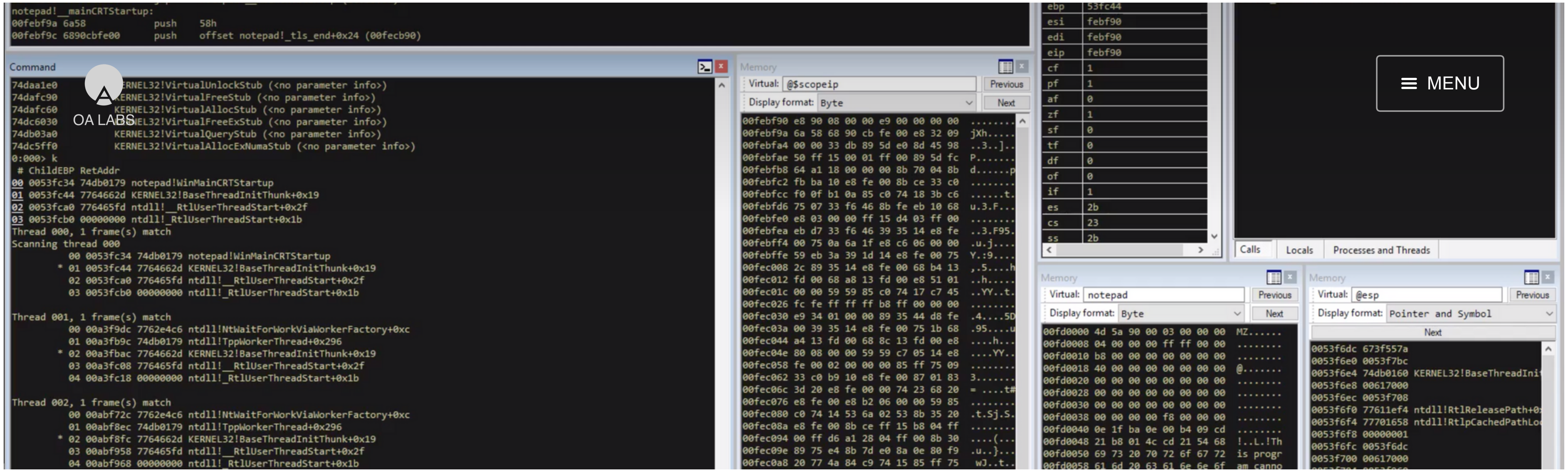


```
notepad!__mainCRTStartup:
00febf9a 6a58            push    58h
00febf9c 6890cbfe00      push    offset notepad!_tls_end+0x24 (00fecb90)
```

Command

```
74daa1e0          ERNEL32!VirtualUnlockStub (<no parameter info>)
74dafc90          KERNEL32!VirtualFreeStub (<no parameter info>)
74dafc60          KERNEL32!VirtualAllocStub (<no parameter info>)
74dc6030            EL32!VirtualFreeExStub (<no parameter info>)
74db03a0          KERNEL32!VirtualQueryStub (<no parameter info>)
74dc5ff0          KERNEL32!VirtualAllocExNumaStub (<no parameter info>)
0:000> k
 # ChildEBP RetAddr
00 0053fc34 74db0179 notepad!WinMainCRTStartup
01 0053fc44 7764662d KERNEL32!BaseThreadInitThunk+0x19
02 0053fca0 776465fd ntdll!__RtlUserThreadStart+0x2f
03 0053fcb0 00000000 ntdll!_RtlUserThreadStart+0x1b
Thread 000, 1 frame(s) match
Scanning thread 000
          00 0053fc34 74db0179 notepad!WinMainCRTStartup
        * 01 0053fc44 7764662d KERNEL32!BaseThreadInitThunk+0x19
          02 0053fca0 776465fd ntdll!__RtlUserThreadStart+0x2f
          03 0053fcb0 00000000 ntdll!_RtlUserThreadStart+0x1b

Thread 001, 1 frame(s) match
          00 00a3f9dc 7762e4c6 ntdll!NtWaitForWorkViaWorkerFactory+0xc
          01 00a3fb9c 74db0179 ntdll!TppWorkerThread+0x296
        * 02 00a3fbac 7764662d KERNEL32!BaseThreadInitThunk+0x19
          03 00a3fc08 776465fd ntdll!__RtlUserThreadStart+0x2f
          04 00a3fc18 00000000 ntdll!_RtlUserThreadStart+0x1b

Thread 002, 1 frame(s) match
          00 00abf72c 7762e4c6 ntdll!NtWaitForWorkViaWorkerFactory+0xc
          01 00abf8ec 74db0179 ntdll!TppWorkerThread+0x296
        * 02 00abf8fc 7764662d KERNEL32!BaseThreadInitThunk+0x19
          03 00abf958 776465fd ntdll!__RtlUserThreadStart+0x2f
          04 00abf968 00000000 ntdll!_RtlUserThreadStart+0x1b
```

Memory

Virtual: @$scopeip

Display format: Byte

```
00febf90 e8 90 08 00 00 e9 00 00 00 00  ........
00febf9a 6a 58 68 90 cb fe 00 e8 32 09  jXh.....
00febfa4 00 00 33 db 89 5d e0 8d 45 98  ..3..]..
00febfae 50 ff 15 00 01 ff 00 89 5d fc  P.......
00febfb8 64 a1 18 00 00 00 8b 70 04 8b  d......p
00febfc2 fb ba 10 e8 fe 00 8b ce 33 c0  ........
00febfcc f0 0f b1 0a 85 c0 74 18 3b c6  ......t.
00febfd6 75 07 33 f6 46 8b fe eb 10 68  u.3.F...
00febfe0 e8 03 00 00 ff 15 d4 03 ff 00  ........
00febfea eb d7 33 f6 46 39 35 14 e8 fe  ..3.F95.
00febff4 00 75 0a 6a 1f e8 c6 06 00 00  .u.j....
00febffe 59 eb 3a 39 1d 14 e8 fe 00 75  Y.:9....
00fec008 2c 89 35 14 e8 fe 00 68 b4 13  ,.5....h
00fec012 fd 00 68 a8 13 fd 00 e8 51 01  ..h.....
00fec01c 00 00 59 59 85 c0 74 17 c7 45  ..YY..t.
00fec026 fc fe ff ff ff b8 ff 00 00 00  ........
00fec030 e9 34 01 00 00 89 35 44 d8 fe  .4....5D
00fec03a 00 39 35 14 e8 fe 00 75 1b 68  .95....u
00fec044 a4 13 fd 00 68 8c 13 fd 00 e8  ....h...
00fec04e 80 08 00 00 59 59 c7 05 14 e8  ....YY..
00fec058 fe 00 02 00 00 00 85 ff 75 09  ........
00fec062 33 c0 b9 10 e8 fe 00 87 01 83  3.......
00fec06c 3d 20 e8 fe 00 00 74 23 68 20  = ....t#
00fec076 e8 fe 00 e8 b2 06 00 00 59 85  ........
00fec080 c0 74 14 53 6a 02 53 8b 35 20  .t.Sj.S.
00fec08a e8 fe 00 8b ce ff 15 b8 04 ff  ........
00fec094 00 ff d6 a1 28 04 ff 00 8b 30  ....(...
00fec09e 89 75 e4 8b 7d e0 8a 0e 80 f9  .u..}...
00fec0a8 20 77 4a 84 c9 74 15 85 ff 75   wJ..t..
```

```
ebp  53fc44
esi  febf90
edi  febf90
eip  febf90
cf   1
pf   1
af   0
zf   1
sf   0
tf   0
df   0
of   0
if   1
es   2b
cs   23
ss   2b
```

Memory

Virtual: notepad

Display format: Byte

```
00fd0000 4d 5a 90 00 03 00 00 00  MZ......
00fd0008 04 00 00 00 ff ff 00 00  ........
00fd0010 b8 00 00 00 00 00 00 00  ........
00fd0018 40 00 00 00 00 00 00 00  @.......
00fd0020 00 00 00 00 00 00 00 00  ........
00fd0028 00 00 00 00 00 00 00 00  ........
00fd0030 00 00 00 00 00 00 00 00  ........
00fd0038 00 00 00 00 f8 00 00 00  ........
00fd0040 0e 1f ba 0e 00 b4 09 cd  ........
00fd0048 21 b8 01 4c cd 21 54 68  !..L.!Th
00fd0050 69 73 20 70 72 6f 67 72  is progr
00fd0058 61 6d 20 63 61 6e 6e 6f  am canno
```

Memory

Virtual: @esp

Display format: Pointer and Symbol

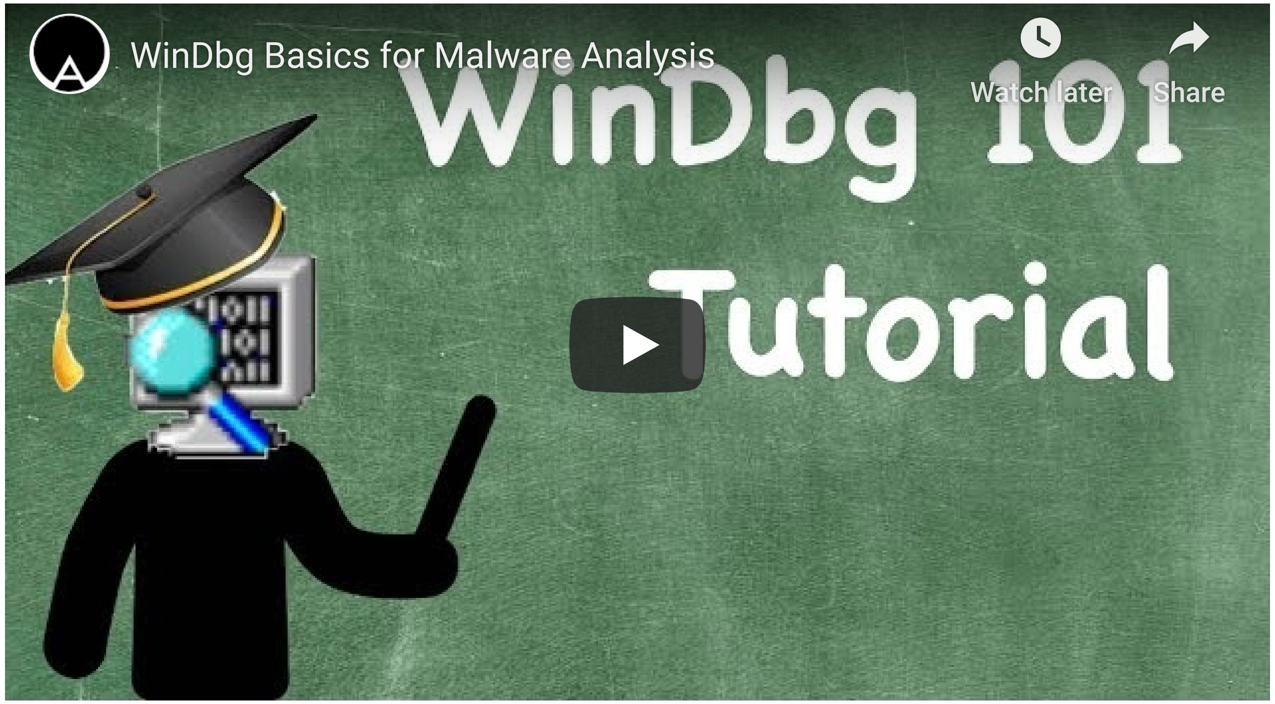
```
0053f6dc 673f557a
0053f6e0 0053f7bc
0053f6e4 74db0160 KERNEL32!BaseThreadInit
0053f6e8 00617000
0053f6ec 0053f708
0053f6f0 77611ef4 ntdll!RtlReleasePath+0
0053f6f4 77701658 ntdll!RtlpCachedPathLo
0053f6f8 00000001
0053f6fc 0053f6dc
0053f700 00617000
```

WinDbg Malware Analysis Cheat Sheet

18 FEBRUARY 2019 on Tutorials

A big thanks to our friend Josh for helping with this post. Go follow him over on Twitter for more excellent reverse engineering content!

Follow @jershmagersh



Workspace Setup

WinDbg workspace customization has been coved in detail by Zach Burlingame over on his site here. To import his custom workspace we have archived the registry key he provides in his post as a gist **windbg.reg**. Simply download the gist and add it to your registry:

- Open `regedit.exe`
- On the **Registry** menu, click **Import Registry File**
- In the Import Registry File dialog box, select the `windbg.reg` file you downloaded, and click **Open**.

Now reopen WinDbg and you should see a nicely formatted workspace. As noted in his Zach's blog post you will need to fix the host specific path information in the workspace after opening it for the first time in WinDbg.

Symbol Setup

In order to take full advantage of WinDbg's features you will need to load the symbols for the DLLs that your malware will load during runtime (or at the very least symbols for `Kernel32.dll`, and `ntdll.dll`).

First you will need to download the symbols to your analysis host using the `symchk.exe` tool. This tool is installed along with WinDbg when you install the Windows SDK.

- Create a directory to store your symbols in. We use `C:\symbols`
- Open `cmd.exe` and navigate to the path where you have installed the Windows SDK.
- Use the following command with **symchk** to download the symbols you need.
  `symchk /r c:\windows\system32\kernel32.dll /s SRV*c:\symbols\*http://msdl.microsoft.com/download/symbols`
- You can replace `c:\windows\system32\kernel32.dll` with the path to the DLL you need to download the symbols for.
  *Remember for 32 bit malware you will need to specify the path to the 32 bit DLLs in the SysWow64 directory. For example, `C:\Windows\SysWOW64\kernel32.dll`

After you have loaded the symbols you need to your symbols directory you will need to set the symbol path in WinDbg and ask it to load the symbols from your directory. This is a two step process using the WinDbg command line.

- First tell WinDbg where to find your symbols directory:
  `.sympath C:\symbols`
- Then force a reload of the symbols:
  `.reload /f`

Pseudo-Regisiter Cheat Sheet

Pseudo-Registers are used to quickly reference registers and other important information while debugging. For example, `$eax` for the eax register, `$scopeip` for the in scope instruction pointer, and `$exentry` for the entrypoint of the executable. They are easily identified by the ($) and (@) symbols.

Microsoft provides a full list of these registers here: **Pseudo-Register Syntax**.

Exploring Memory

!address
List all memory segments in the process with their permissions and memory type. This is similar to the **Memory Map** in x64dbg.

!address [virtual address]
List information about the memory segment that contains the virtual address.
For example `!address 400000` would show information about the section that contains address 0x40000. This is a quick way to find the start and end of a memory segment if you want to dump it.

.writemem [file name] [start address] [end address]
Dump memory range to file.
For example, `.writemem C:\dump.bin 400000 401000` would dump memory starting at 0x40000 and ending at 0x401000 to the dump.bin file.

eb [address] [byte]
Enter one byte into memory at the address.
For example, `eb 400000 0xff` would change the byte at address 0x400000 to 0xff.
For a full list of enter commands (string, word, etc.) see the Microsoft docs **here**.

Loaded Modules (DLLs)

lm
List loaded modules (show all the DLLs).

lm -m [pattern]
List loaded modules that match the pattern.
For example, `lm -m kernel*` would find and list `kernel32.dll` if it had been loaded into the process space.

lm -a [address]
List the module loaded at that address.
For example, `lm -a 400000` would show the module that is loaded at the address 0x400000.

ln [address]
List the nearest symbol to the address. This is a good way to figure out what is near an address if you get lost debugging.

x [module!symbol]
Examine the symbols for a given module.
For example, `x kernel32!*` will list all the symbols for `kernel32.dll`.

Debugging

g
Go! This will run the debugger.

bp [address]
Add a breakpoint at the address. You can also use pseudo-registers, and symbols instead of address.

For example, `bp $exentry` will add a breakpoint to the entry point of the PE that spawned the process. And `bp Kernel32!GlobalAlloc` will add a breakpoint on the GlobalAlloc API.

bu [address]
Adds an unresolved breakpoint. This differs from a standard breakpoint as it is not set on the specific address and can be set on addresses that do not resolve (because a module has not yet been loaded).
If you use `bp` to set a breakpoint on an address that doesn't resolve it will automatically be converted into a `bu`.
See this Microsoft explanation on **Unresolved Breakpoints**.

bl
List all breakpoints.

bc [number]
Clear the breakpoint at that number in the breakpoint list (list them first).
You can also use `*` to clear all breakpoints.

Data Types and Structures

dt [module!symbol] [address]
List the address a type cast to the symbol specified. This is very useful for examining structures that are returned from API calls.
For example `dt -r ntdll!_IMAGE_NT_HEADERS 400100` will display the address at 0x400100 as the struct `_IMAGE_NT_HEADERS` recursively (also casting all memory pointed to by the struct).

Further Reading...

Check out this more **in-depth cheat** sheet from Devon Greene, and his nice set of **WinDbg debugging tricks**.

Talos tutorial for using the **JavaScript extension with newer versions of WinDbg**.

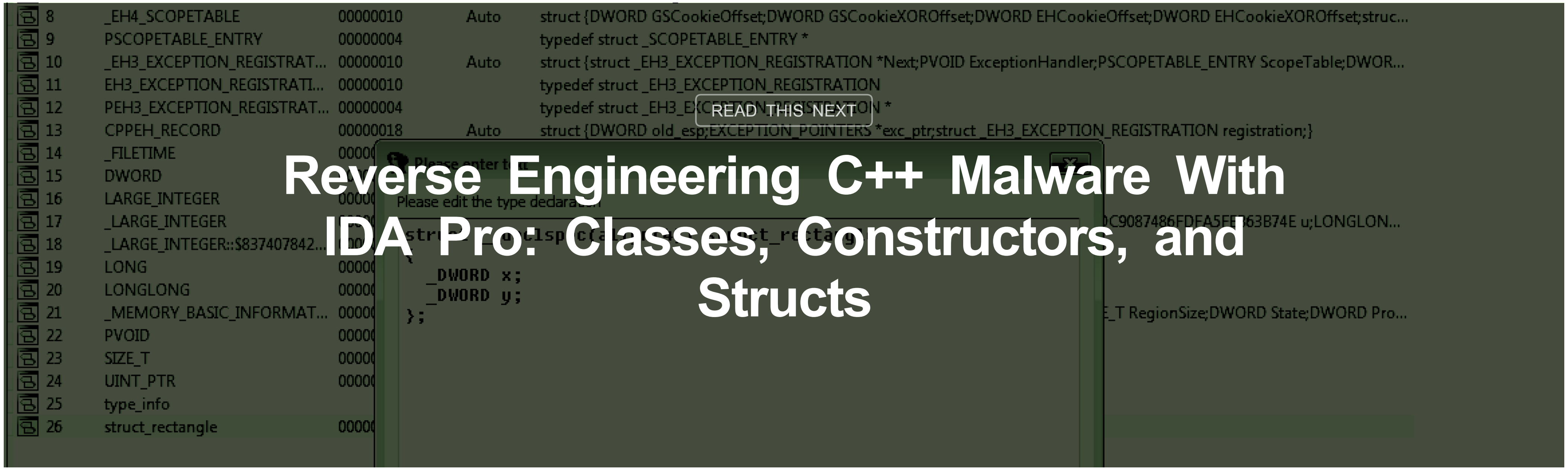
Microsoft full **list of commands** which is terribly formatted!

Sergei Frankoff

Sergei a co-founder of Open Analysis, and volunteers as a malware researcher. His focus is reverse engineering malware and building automation tools for malware analysis.

https://www.openanalysis.net

Share this post



Reverse Engineering C++ Malware With IDA Pro: Classes, Constructors, and Structs

YOU MIGHT ENJOY

OALabs Malware Analysis Virtual Machine