



# Trick-or-treat WinDbg: taming the Windows debugger

44con 2024 - Workshop

# Today's goal

How to make the most of a debugger in a minimum amount of time





1

Introduction to Windbg

2

Tips to navigate Windbg

3

The secret weapon: DrawMeATree

4

Time for some trick-or-treat!

5

Conclusion



# @MathildeVenault

- > Security Researcher at CrowdStrike
- > Ex-volunteer firefighter
- > Like to share my work in blog posts & conferences (Black Hat USA, REcon, c0c0n ...)
- > Love baking!





# Introduction to WinDbg



# Why/when using a debugger

## > Dynamic analysis:

- **Understand** a behavior (eg: malware analysis)
- **Identify** variables (eg: RE undocumented components)
- Change execution flow (eg: bypass evasion attempts)

> vs the static analysis that gives a “frozen view” and where you have limited interactions



# Setting up a debugging session

The screenshot shows the WinDbg 1.2402.24001.0 application window. The 'Start debugging' menu is open, displaying various options. Three red arrows point from text annotations to specific menu items:

- Start a target binary** points to **Launch executable (advanced)** (Supports Time Travel Debugging).
- Attach to a running process** points to **Attach to process** (Supports Time Travel Debugging).
- Attach to a remote system (ie: VM)** points to **Attach to kernel**.

The right pane shows the configuration for the selected option, with fields for Executable, Arguments, Start directory, Target architecture (set to Autodetect), and checkboxes for Debug child processes and Record with Time Travel Debugging.



# Interacting with Windbg

“Execute until breakpoint”

“Restart from the beginning of the execution”

“Pause execution here”

Main window

Interactive interface to enter commands

The screenshot shows the Windbg interface with several annotations:

- A red arrow points to the **Break** button (two vertical bars) with the text “Pause execution here”.
- A red arrow points to the **Step Out** button (a green play button with a bracket) with the text “Execute until breakpoint”.
- A red arrow points to the **Restart** button (a green circular arrow) with the text “Restart from the beginning of the execution”.
- A red arrow points to the **Command** window at the bottom with the text “Interactive interface to enter commands”.
- A red arrow points to the main window area showing the **ModLoad** list with the text “Main window”.

The **Command** window displays the following output:

```
Disassembly Registers Memory 0
Command X
ModLoad: 77310000 77400000 C:\Windows\SysWOW64\KERNEL32.DLL
ModLoad: 75e20000 7605a000 C:\Windows\SysWOW64\KERNELBASE.dll
ModLoad: 751a0000 75240000 C:\Windows\SysWOW64\apphelp.dll
ModLoad: 76d50000 77306000 C:\Windows\SysWOW64\SHELL32.dll
ModLoad: 73b00000 73b08000 C:\Windows\SysWOW64\VERSION.dll
ModLoad: 759e0000 75a5b000 C:\Windows\SysWOW64\msvc_p_win.dll
ModLoad: 760b0000 7616f000 C:\Windows\SysWOW64\msvcrt.dll
ModLoad: 76170000 76290000 C:\Windows\SysWOW64\ucrtbase.dll
ModLoad: 75510000 756ac000 C:\Windows\SysWOW64\USER32.dll
ModLoad: 76d30000 76d48000 C:\Windows\SysWOW64\win32u.dll
ModLoad: 76200000 762b4000 C:\Windows\SysWOW64\GDI32.dll
```

The command prompt at the bottom shows `0:000>`.





# Setting the initial breakpoint

Command	Use
<code>bp XX</code>	Set a breakpoint at XX
<code>bp \$exentry</code>	Set a breakpoint at the entry point
<code>bl</code>	List breakpoints
<code>bc X   bd X</code>	Clear   disable breakpoint index X
<code>? module!func*</code>	Check if the address of a target function is “resolved”
<code>lm</code>	List Modules (shows loaded modules' info )



# Executing through

Command	Use
k	Display the function call stack
p	Step over the next instruction
pc	Step until next call
pt	Step until next return
t	Step into the next instruction
g	Go: execute until a breakpoint/stop



# Displaying variables

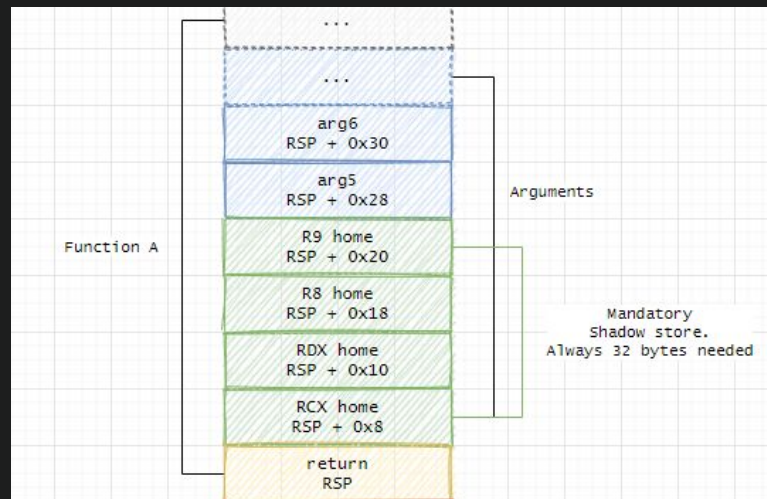
Command	Use
r	“Display <b>r</b> egisters”: shows content of major registers
? XX	“Show what X contains”
d (b   w   d   q)	“Display <b>b</b> yte ... <b>q</b> word X”: display X as bytes ...qword



# X64 calling convention

> (really) long story short:

- Return value = `rax`
- 1st argument = `rcx`
- 2nd argument = `rdx`
- 3rd argument = `r8`
- 4th argument = `r9`
- 5th argument = `rsp + 0x28`
- 6th argument = `rsp + 0x28 + 4`  
= `rsp + 0x30`



From [www.ired.team](http://www.ired.team)



# Miscellaneous

Command	Use
<code>.cls</code>	Clear screen
<code>.formats X</code>	Display x under different <b>formats</b> (hex, dec, ascii)
<code>.logopen C:\Path\.txt /.logclose</code>	<b>Logs</b> the debugging session into the file
<code>.childbdg 1</code>	Enable <b>child</b> process <b>debugging</b>





# Tips to navigate WinDbg



# Searching for symbols

> Search in symbols: `x mymodule!*`

```
0:000> x cmd!*main*
00007ff6`9476d4f8 cmd!main (void)
00007ff6`9477f760 cmd!__srt_common_main_seh (void)
00007ff6`94785450 cmd!`__srt_common_main_seh'::`1'::filt$0 (void)
00007ff6`9477f8f0 cmd!mainCRTStartup (mainCRTStartup)
00007ff6`947aca28 cmd!hMainThread = <no type information>
00007ff6`947b3020 cmd!MainEnv = <no type information>
```



# Using extensions

> Display the last error: `!gle (last error)`

```
0:000> !gle
```

```
LastErrorValue: (Win32) 0x490 (1168) - Element not found.
```

```
LastStatusValue: (NTSTATUS) 0xc000007c - An attempt was made to reference a token that doesn't exist.
```





# Using extensions

> Display structures: !handle, !sid, !process, !thread, !peb, etc...

```
0:000> !peb
PEB at 0000000f1c03f000
  InheritedAddressSpace: No
  ReadImageFileExecOptions: No
  BeingDebugged: Yes
  ImageBaseAddress: 00007ff7cca30000
  NtGlobalFlag: 70
  NtGlobalFlag2: 0
  Ldr: 00007ffb572b6440
  Ldr.Initialized: Yes
  Ldr.InInitializationOrderModuleList: 00000121a3a14010 . 00000121a3a14780
  Ldr.InLoadOrderModuleList: 00000121a3a141e0 . 00000121a3a17b10
  Ldr.InMemoryOrderModuleList: 00000121a3a141f0 . 00000121a3a17b20
      Base TimeStamp      Module
7ff7cca30000 d557d400 Jun 03 19:48:32 2083 C:\Windows\system32\cmd.exe
7ffb57130000 67ca8829 Mar 06 21:46:17 2025 C:\Windows\SYSTEM32\ntdll.dll
```



# Displaying structures

> Display a known structure: `dt dll!_structure`

```
0:000> db 000002b2`b3903720
000002b2`b3903720 36 00 38 00 00 00 00 00-30 3d 90 b3 b2 02 00 00 6.8.....0=.....
000002b2`b3903730 1e 00 20 00 00 00 00 00-68 3d 90 b3 b2 02 00 00 .. .....h=.....
000002b2`b3903740 00 00 02 00 00 00 00 00-88 3d 90 b3 b2 02 00 00 .....=.....
000002b2`b3903750 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
000002b2`b3903760 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
000002b2`b3903770 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
000002b2`b3903780 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
000002b2`b3903790 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
0:000> dt nt!_UNICODE_STRING 000002b2`b3903720
ntdll!_UNICODE_STRING
"C:\Windows\System32\cmd.exe"
+0x000 Length : 0x36
+0x002 MaximumLength : 0x38
+0x008 Buffer : 0x000002b2`b3903d30 "C:\Windows\System32\cmd.exe"
```



# Fast and furious

> Trace execution: `wt (opt) -l <depth to filter>`

```
cmd!main:
00007ff7`1284d4f8 48895c2408      mov     qword ptr [rsp+8],rbx
0:000> wt -l 8
Tracing cmd!main to return address 00007ff7`1285f876
  14      0 [ 0] cmd!main
   3      0 [ 1]  KERNEL32!GetCurrentThreadId
  19      3 [ 0] cmd!main
   1      0 [ 1]  KERNEL32!OpenThreadStub
  19      0 [ 1]  KERNELBASE!OpenThread
   6      0 [ 2]    ntdll!NtOpenThread
>> More than one level popped 1 -> 1
  25      6 [ 1]  KERNELBASE!OpenThread
  22     35 [ 0] cmd!main
   9      0 [ 1]  cmd!CmdSetThreadUILanguage
   8      0 [ 2]    KERNELBASE!GetModuleHandleW
  53      0 [ 3]    ntdll!RtlInitUnicodeString
```





DrawMeATree



# The problem we had

> wt's has answers to our questions, but its results can be hard to exploit

```
0:000> wt
```

```
Tracing sc!wmain to return address 00007ff7`77aa2136
```

6	0	[ 0]	sc!wmain	23008	10	19696	[ 3]	RPCRT4!NdrClientCall2
32	0	[ 1]	sc!Worker	23009	28	19797	[ 2]	sechost!CloseServiceHandle
12	0	[ 2]	KERNELBASE!SetThrea	23010	226	872441	[ 1]	sc!Worker
20	0	[ 3]	KERNELBASE!ResetT	23011	12	0	[ 2]	KERNELBASE!LocalFree
64	0	[ 4]	ntdll!RtlGetThre	23012	15	0	[ 3]	ntdll!RtlFreeHeap
11	0	[ 5]	ntdll!RtlpCre	23013	23	15	[ 2]	KERNELBASE!LocalFree
12	0	[ 6]	ntdll!RtlpI	23014	231	872479	[ 1]	sc!Worker
54	0	[ 7]	ntdll!Rtl	23015	6	0	[ 2]	sc!_security_check_cookie
6	0	[ 8]	ntdll!	23016	241	872485	[ 1]	sc!Worker
57	6	[ 7]	ntdll!Rtl	23017	12	872726	[ 0]	sc!wmain
20	63	[ 6]	ntdll!RtlpI	23018				
13	83	[ 5]	ntdll!RtlpCre	23019	872738 instructions were executed in 872737 events (0 from other threads)			
10	0	[ 6]	ntdll!RtlEn	23020				
17	93	[ 5]	ntdll!RtlpCre	23021	Function Name			
19	0	[ 6]	ntdll!RtlpM	23022	Invocations MinInst MaxInst AvgInst			
					1 2 2 2			
					KERNEL32!BasepNotifyLoadStringResource			



# Our solution: DrawMeATree

- > Python tool to convert wt's output into customizable graphic trees
- > Use cases:
  - Identify components, functionalities, overall operation
  - Highlight connections between functions and modules
  - Summarize a large and complex amount of information with custom filters



# DrawMeATree: Interface

## > Command line interface:

```
C:\Users\User\Desktop\drawmeatree>python draw.py C:\logs\wt_output.txt
```



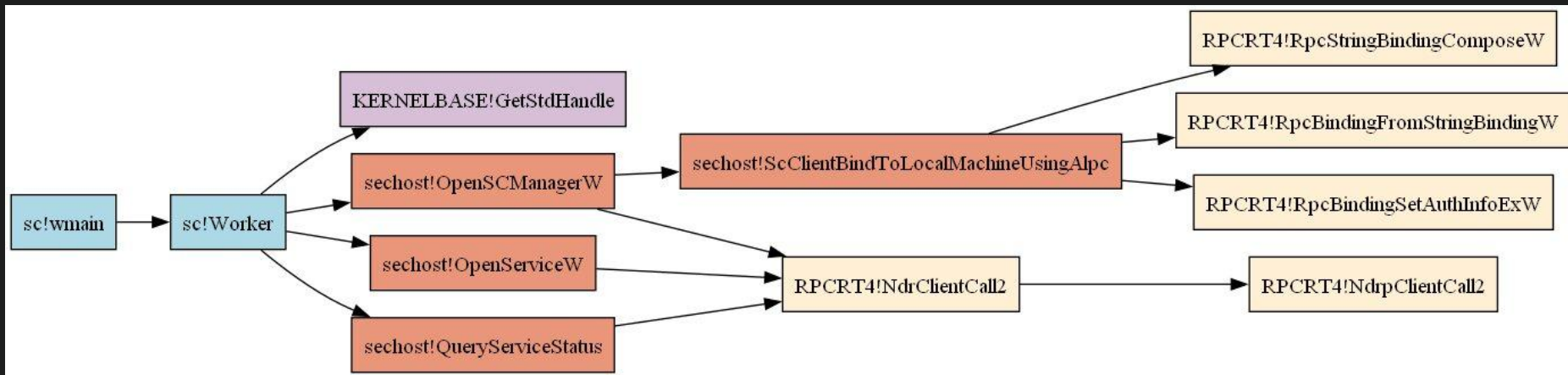
```
[16:34:45] INFO > Processing wt result with the following parameters: draw.py:405
|_ Input file: C:\logs\wt_output.txt
|_ Depth level: 9
|_ Filter words: ['CriticalSection', 'security_check', 'Alloc', 'Heap', 'free',
'operator', 'LockExclusive', 'Error', 'mkstr', 'toupper', 'tolower', 'Unicode', 'tolower',
'toupper']
|_ Print in terminal: False
|_ Direction of trees: LR
|_ Export results in: C:\Users\User\Desktop\drawmeatree

INFO > Parsing of wt output... OK draw.py:408
INFO > Creation of the trees... OK draw.py:412
[16:34:58] INFO > Generation of the pngs... OK draw.py:418
INFO [+] Success! Trees have been generated in: draw.py:423
C:\Users\User\Desktop\drawmeatree.
```



# DrawMeATree: filtered tree example

> Resulting filtered tree from the previous wt's output:





# DrawMeATree: limits

## > Limits:

- Functions appear **without a chronological order**
- Windows' default image viewer sometimes is too slow to load, need to use another software (eg: JPEGview)
- Dependency "dot" is limits a **number of lines** (around 80 000 lines) to generate the trees



# DrawMeATree: filtering methodology

1. **Generate** your wt into a log file
2. Identify the **maximum depth** required to filter it (or regenerate from a function of interest if too big)
3. Remove **irrelevant “high level” node** that will clear entire branches (allocation/freeing phases, default routines, string handling)
4. Remove nodes/modules that **do not answer your initial question**



# Practice time

> Question:



**How does “whoami.exe” find the current user name?**

*Found it? Try to set a bp at the key moment just before the programs retrieves the info, to verify your assumption.*



# Practice time

> Question:

How does “taskkill.exe” terminate a target process?



*Basic usage of taskkill can be:*

```
C:\Users\User>taskkill /im notepad.exe  
SUCCESS: Sent termination signal to the process "Notepad.exe" with PID 1456.
```

*Found it? Try to find a way to change the target process killed by taskkill.exe :)*



# Practice time



> Question

How does “mklink.exe” create a symbolic link?





# Conclusion



# General methodology for debugging

- > 1/ Start the debugging at a chosen point (bp \$exentry, bp module!FunctionX, etc.)
- > 2/ If you don't have a target -> use wt + DrawMeATree to get a better overview of the execution flow
- > 3/ Get as close as possible to your target & observe behaviors



# What's next?

- > Practice, struggle, and then practice again
- > Learn advanced methodology with Yarden Shafir's work





# References

> MSDN official documentation:

<https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger>

> Cheat sheet:

<http://windbg.info/doc/1-common-cmds.html>

> Yarden Shafir's work:

<https://medium.com/@yardenshafir2/windbg-the-fun-way-part-1-2e4978791f9b>

[https://github.com/yardenshafir/WinDbg\\_Scripts](https://github.com/yardenshafir/WinDbg_Scripts)

> Kitty images: <https://www.freepik.com/>



# Bonus: conditional breakpoints

> “Break on access”:      `ba w|r 4 XX`

```

UseCase0!sscanf_s:
00007ff7`b4291180 4889542410      mov     qword ptr [rsp+10h],rdx ss:0
0:000> ? r8
Evaluate expression: 928674478320 = 000000d8`394ff8f0
0:000> pt
UseCase0!sscanf_s+0x67:
00007ff7`b42911e7 c3             ret
0:000> ba r4 000000d8`394ff8f0
0:000> g
Breakpoint 4 hit
ucrtbase!_mbcmp_l+0x4a:
00007ffb`627e8e5a 3a043b        cmp     al,byte ptr [rbx+rdi] ds:00

```

Step 1. We wonder where is used an interesting value (ie: r8 here)

Step 2. We wait for our function to finish initializing the value

Step 3. We set up the break on access

Step 4. We wait until the address gets read!

