Trick-or-treat WinDbg: taming the Windows debugger

44con 2024 - Workshop

CROWDSTRIKE

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

CROWDSTRIKE

Why/when using a debugger

> Dynamic analysis:

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

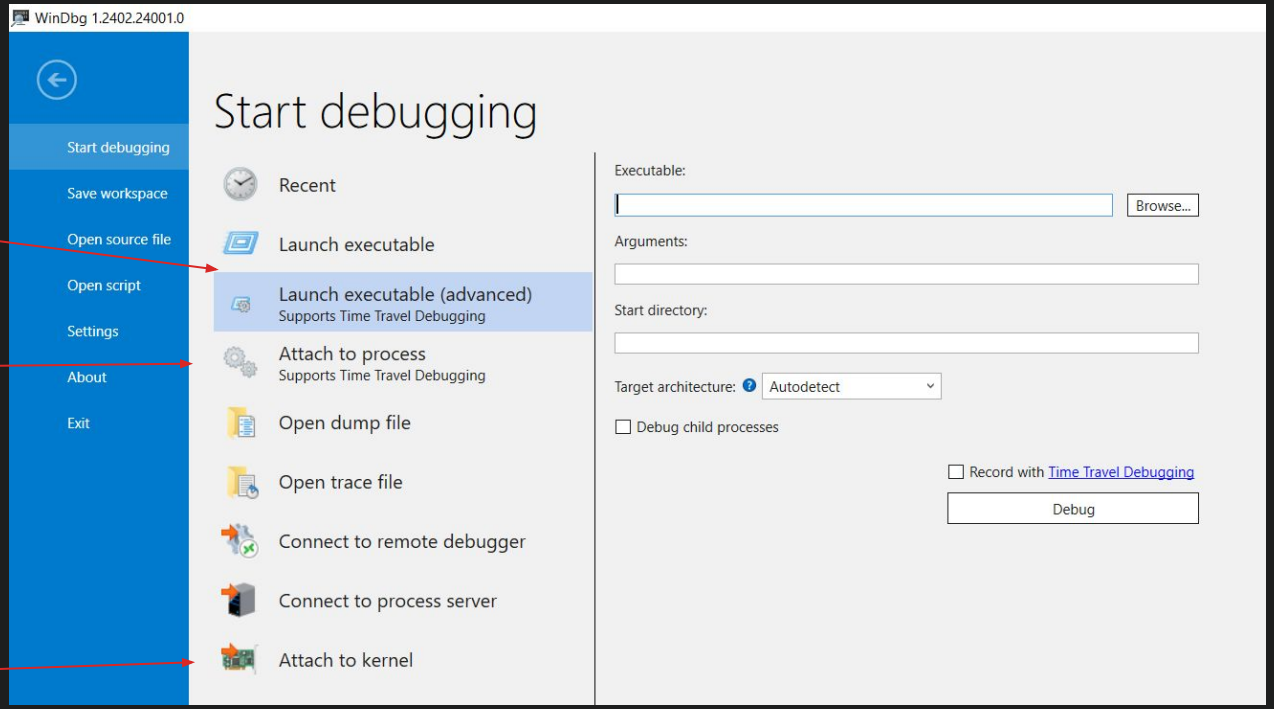
> vs the static analysis that gives a “frozen view” and where you have limited interactions

Setting up a debugging session

Start a target binary

Attach to a running process

Attach to a remote system (ie: VM)

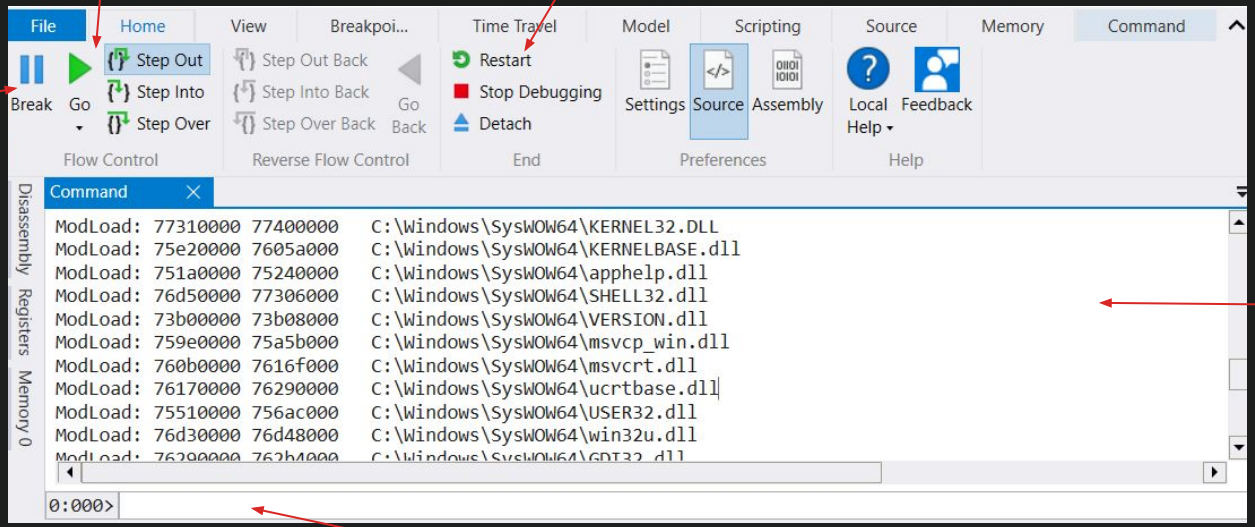


Interacting with Windbg

"Execute until breakpoint"

"Restart from the beginning of the execution"

"Pause execution here"



Main window

Interactive interface to enter commands

Setting the initial breakpoint

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

CROWDSTRIKE

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

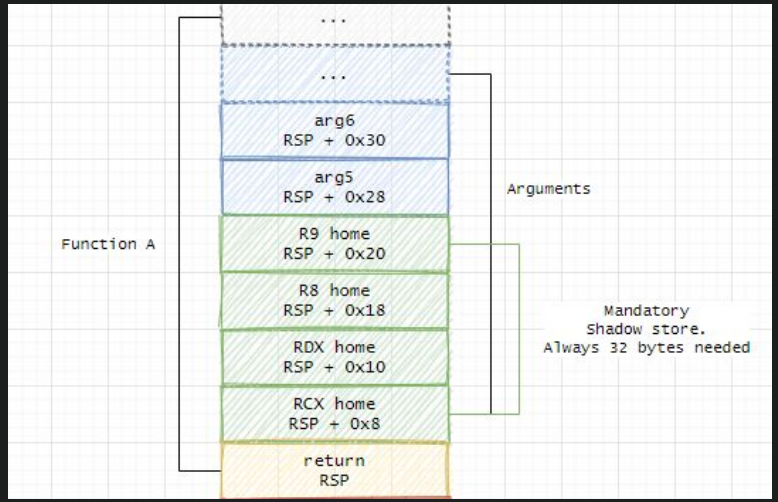
Displaying variables

Command	Use
`r`	“Display registers”: shows content of major registers
`? XX`	“Show what X contains”
`d(b|w|d|q)`	“Display byte|…qword 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

CROWDSTRIKE

Miscellaneous

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

Tips to navigate WinDbg

CROWDSTRIKE

Searching for symbols

> Search in symbols: `x mymodule!*`

```
0:000> x cmd!*main*
00007ff6`9476d4f8 cmd!main (void)
00007ff6`9477f760 cmd!__scrt_common_main_seh (void)
00007ff6`94785450 cmd!`__scrt_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>
```

CROWDSTRIKE

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.
```

CROWDSTRIKE

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
```

The next level: displaying variables

> 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 00  ................
000002b2`b3903760  00 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 00  ................
000002b2`b3903780  00 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 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"
```

The next level: get an overview

> 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
   32     0 [  1]   sc!Worker
   12     0 [  2]     KERNELBASE!SetThrea
   20     0 [  3]       KERNELBASE!ResetT
   64     0 [  4]         ntdll!RtlGetThr
   11     0 [  5]           ntdll!RtlpCre
   12     0 [  6]             ntdll!RtlpI
   54     0 [  7]               ntdll!Rtl
    6     0 [  8]                 ntdll!_
   57     6 [  7]               ntdll!Rtl
   20    63 [  6]             ntdll!RtlpI
   13    83 [  5]           ntdll!RtlpCre
   10     0 [  6]             ntdll!RtlEn
   17    93 [  5]           ntdll!RtlpCre
   19     0 [  6]             ntdll!RtlpM
```

```
23008     10 19696 [  3]       RPCRT4!NdrClientCall2
23009     28 19797 [  2]     sechost!CloseServiceHandle
23010    226 872441 [  1]   sc!Worker
23011     12     0 [  2]     KERNELBASE!LocalFree
23012     15     0 [  3]       ntdll!RtlFreeHeap
23013     23    15 [  2]     KERNELBASE!LocalFree
23014    231 872479 [  1]   sc!Worker
23015      6     0 [  2]     sc!_security_check_cookie
23016    241 872485 [  1]   sc!Worker
23017     12 872726 [  0] sc!wmain
23018
23019  872738 instructions were executed in 872737 events (0 from other threads)
23020
23021  Function Name                               Invocations MinInst MaxInst AvgInst
23022  KERNEL32!BasepNotifyLoadStringResource                1       2       2       2
```

CROWDSTRIKE

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

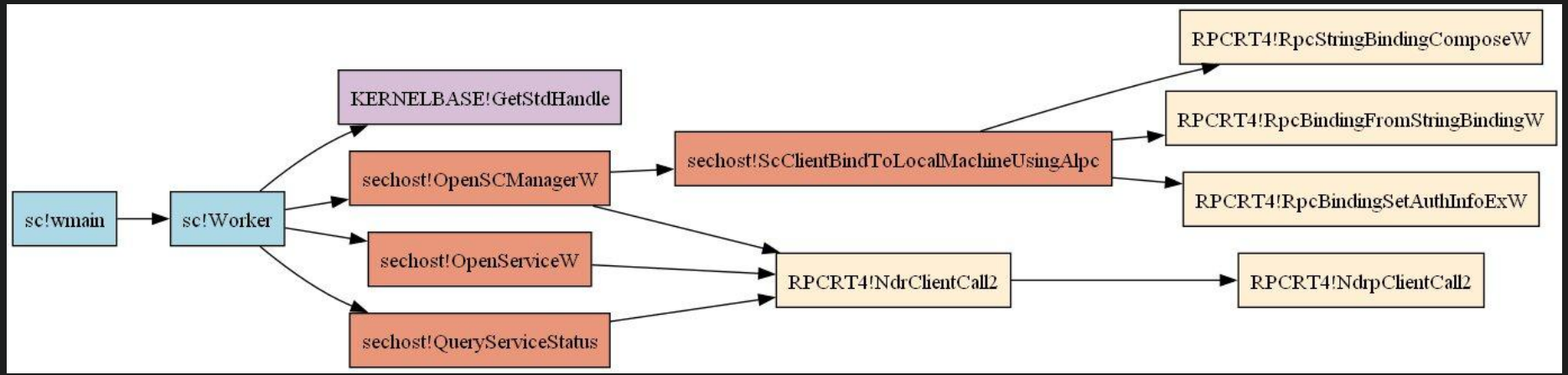
> Command line interface:

```
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', 'towlower',
                    'towupper']
                    |_ 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.
```

CROWDSTRIKE

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

CROWDSTRIKE

Practice time

> Question:

How does “whoami.exe” finds the current domain + 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 :)

CROWDSTRIKE

Practice time

> Question

How does “mklink.exe” creates symbolic link?

Conclusion

CROWDSTRIKE

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

CROWDSTRIKE

What's next?

> Practice, struggle, and then practice again

> Learn advanced methodology with Yarden Shafir's work

CROWDSTRIKE

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

> 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:
0:000> ? 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!_mbscmp_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!