Manual kernel mode analysis with WinDbg

VB2018

Vanja Svajcer

@vanjasvajcer

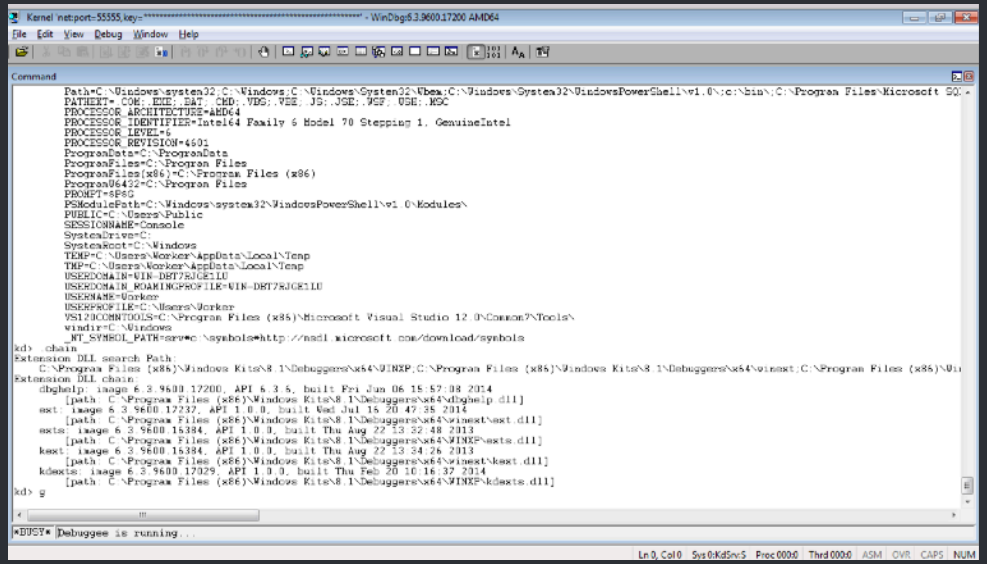
Manual kernel mode analysis with WinDbg

- Intro to WinDbg
  - Setup
  - Basic commands
  - Taking it to the next level
  - Scripting
  - Extensions
- Malware analysis tips

Setting the scene

What is it

- GUI on top of DbgEng in Windows
  - ntsd, csd
  - kd

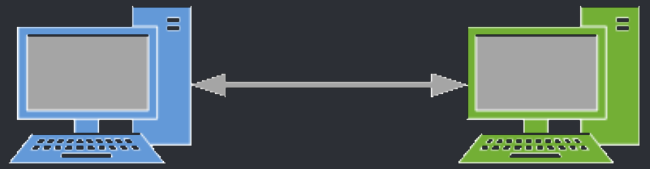


Installation and setup

- Debugging tools for Windows
  - Part of WDK
  - Part of SDK install
  - Microsoft Store

Live debugging setup

- Interfaces:
  - Serial (slow)
  - Firewire (1392)
  - USB
  - Network (TCP/IP)

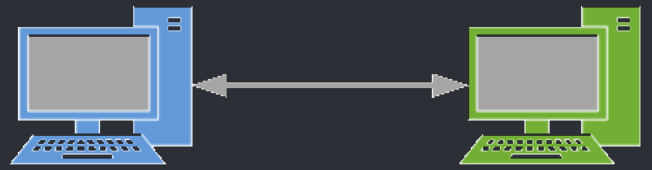


Debugger Host

Debuggee Target

Live debugging setup - VM to VM

- Serial
- Network
- VirtualKD (VMM host to VM only)



Debugger
VM

Debuggee
Target VM

Setup debugging over network

1. find debugger's ip v4 address
2. choose any TCP port (e.g 55555)
3. on the debugee

```
bcdedit.exe -set loadoptions DISABLE_INTEGRITY_CHECKS
bcdedit.exe -set TESTSIGNING ON

bcdedit /debug on

bcdedit /dbgsettings net hostip:w.x.y.z port:n key:xxxx
```

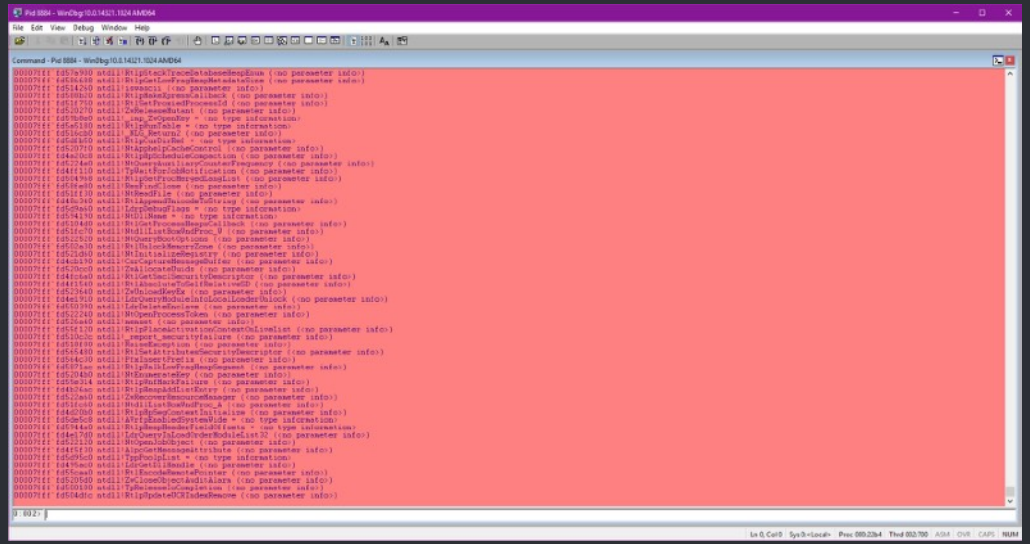
TALOS

Start debugging

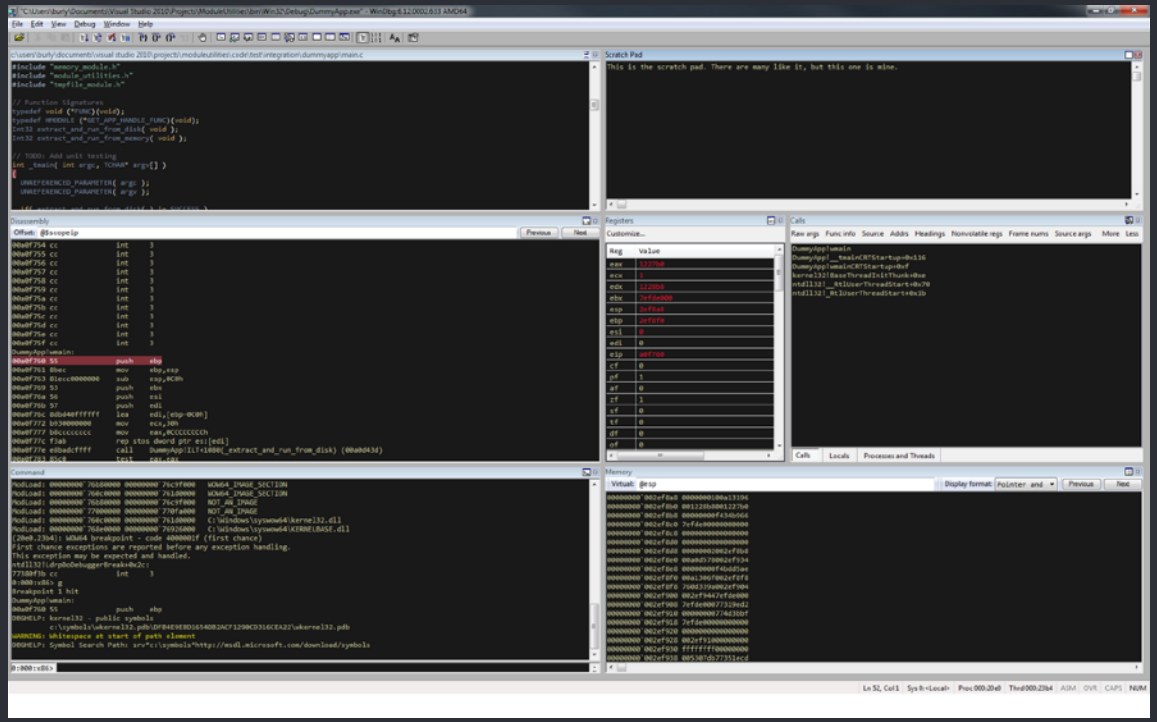
1. Start the debugger
    A. windbg -k net:port=n,key=Key
    B. From GUI: File->Kernel Debug
2. Reboot the debugee
3. PROFIT!

WinDbg Workspaces

- Setup ideal workspace
- Themes as registry values (can be moved by exporting into a registry file)
- Stored in HKCU\Software\Microsoft\Windbg\Workspaces



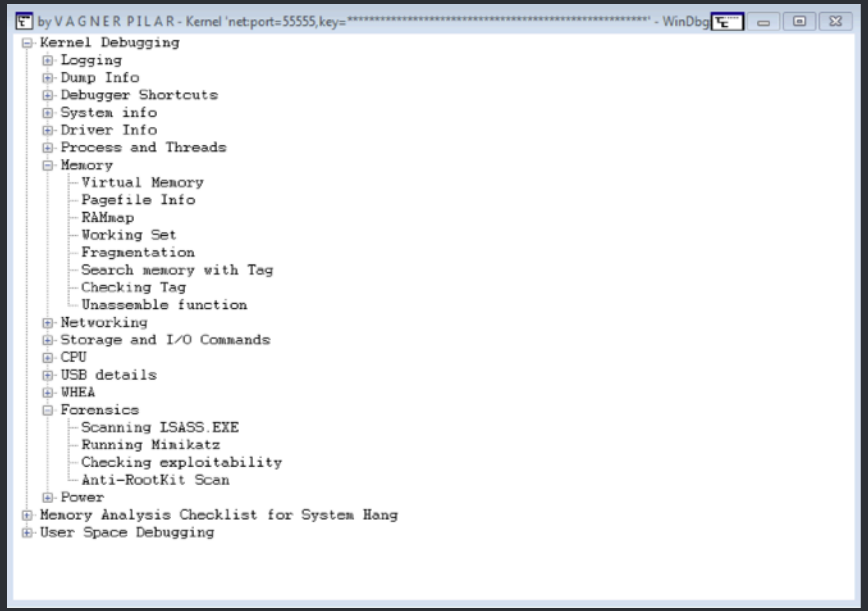
WinDbg Workspaces



http://www.zachburlingame.com/wp-content/uploads/fec46d7fc976_C835/windbg.reg

WinDbg Workspaces

- .cmdtree - useful for learning and remembering commands



- https://github.com/vagnerpilar/windbgtree

TALOS

Downloading Symbols

- _NT_SYMBOL_PATH - environment variable
  - _NT_SYMBOL_PATH=SRV*c:\MyServerSymbols*http://msdl.microsoft.com/download/symbols
- GUI
  - srv*c:\MyServerSymbols*https://msdl.microsoft.com/download/symbols
- Command window
  - .sympath srv*c:\MyServerSymbols*https://msdl.microsoft.com/download/symbols

Basic WinDbg

Logging

- .logopen filepath
- .logclose
- Set verbose mode
- .hh - open help file

Registers and PseudoRegisters

- r vs r?
- r register flags/mask (rM)
- $t0 to $t19
- $csp, $ip
- $ra, $extret, $retreg
- $peb, $teb
- $proc, $thread
- $iment (operator)
- $extret

Exploration commands

- x
- dt
- db, dw, dd, dq, dps, du, da
- k
- ln - where is this?
- !dh - display pe header
- !ustr
- s

Exploration commands

- dx - Explore debugger object model

Disassembling

- u
- uf

Control

- t [address] - trace (Step into)
- p [address] - proceed (Step over)
- pc (tc) - Step over until a call instruction is encountered
- pt (tt) - Step over until return
- g -
- gu - go up (return to the calling function and stop - careful here)
- .process - set process context
- .thread - set register context

Breakpoints

- ba (hardware if possible)
- bp[ID] [Options] [Address [Passes]] [“CommandString"]
- bu (unresolved)
- bm (multiple)

- bl
- .bpcmds
- bc

Breakpoints

- Conditional
- `bp Address "j (Condition) 'OptionalCommands'; 'gc' "`
- `bp Address ".if (Condition) {OptionalCommands} .else {gc}"`
- `bp kernel32!CreateEventW "$$<c:\\commands.txt"`

Exceptions

- sxe ld - break on module load
- sxe cpr - break on process creation
- sx - show all events/exceptions and their statuses

Output

- .printf
- .echo

It is all easy now

Expression Evaluators

- .expr - checking and changing
- ?
- ??
- @@masm, @@c++, @@
- when evaluating a reg @sign is required eg. @$retreg (for all (pseudo) registers)

Pointer dereferencing

- poi(rax)
- da @@c++(((nt!_EPROCESS *) @$proc)->ImageFileName)
- dwo
- qwo

Lists

- dt nt!_LIST_ENTRY
- +0x000 Flink : Ptr64 _LIST_ENTRY
- +0x008 Blink : Ptr64 _LIST_ENTRY

- #CONTAINING_RECORD
- #FIELD_OFFSET

Lists

- Walk a list

```
!list -x "dt nt!_LDR_DATA_TABLE_ENTRY @$extret" @@(&@$peb->Ldr->InLoadOrderModuleList)

!list -x "dt nt!_LDR_DATA_TABLE_ENTRY @$extret BaseDllName DllBAse" nt!PsLoadedModuleList
```

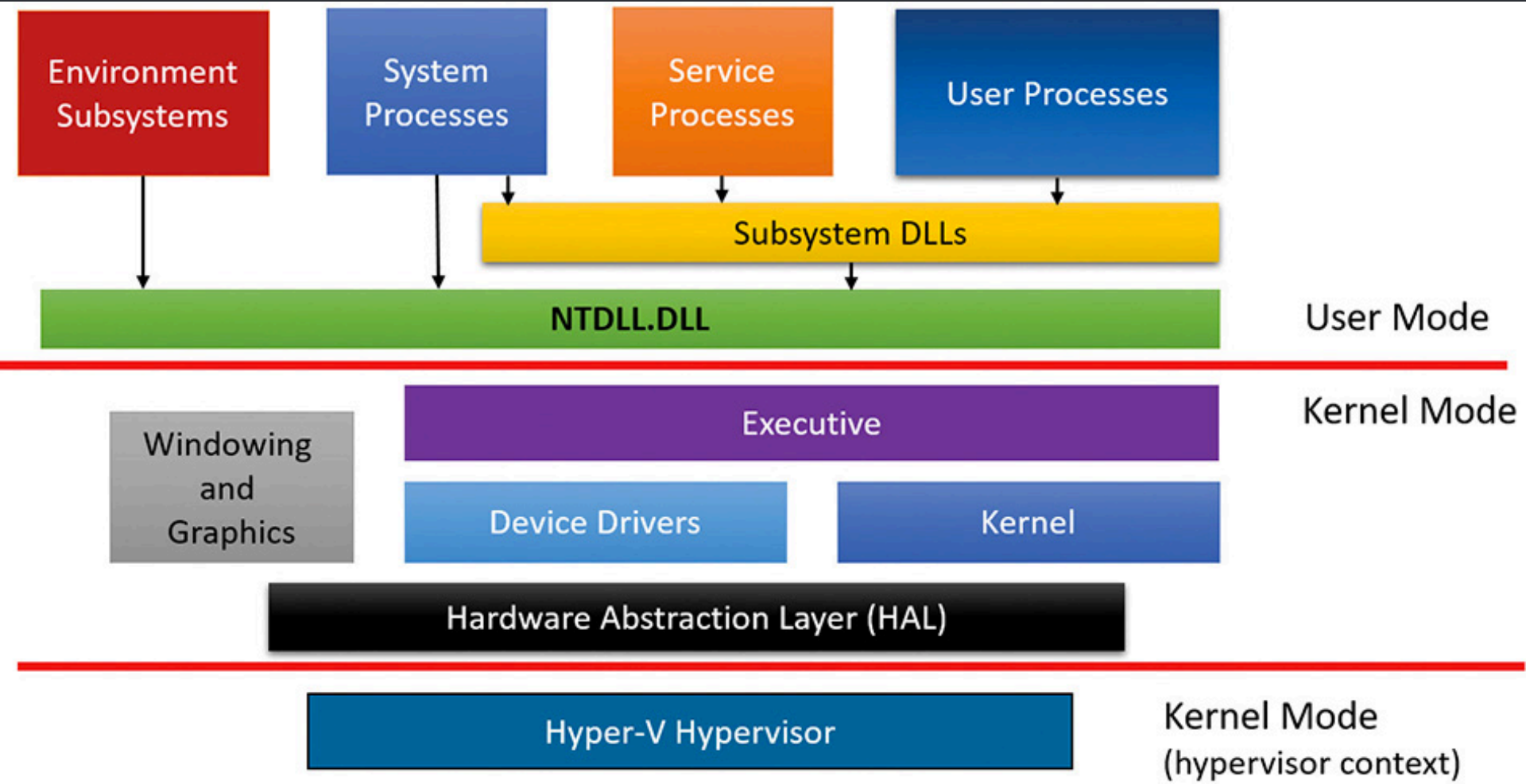
Debugger markup language (DML)

- .dml_start
- .prefer_dml 1|0
- Commands with /D switches
- !dml_proc
- lmD - lm with DML as a result
- .dml_flow Start Target

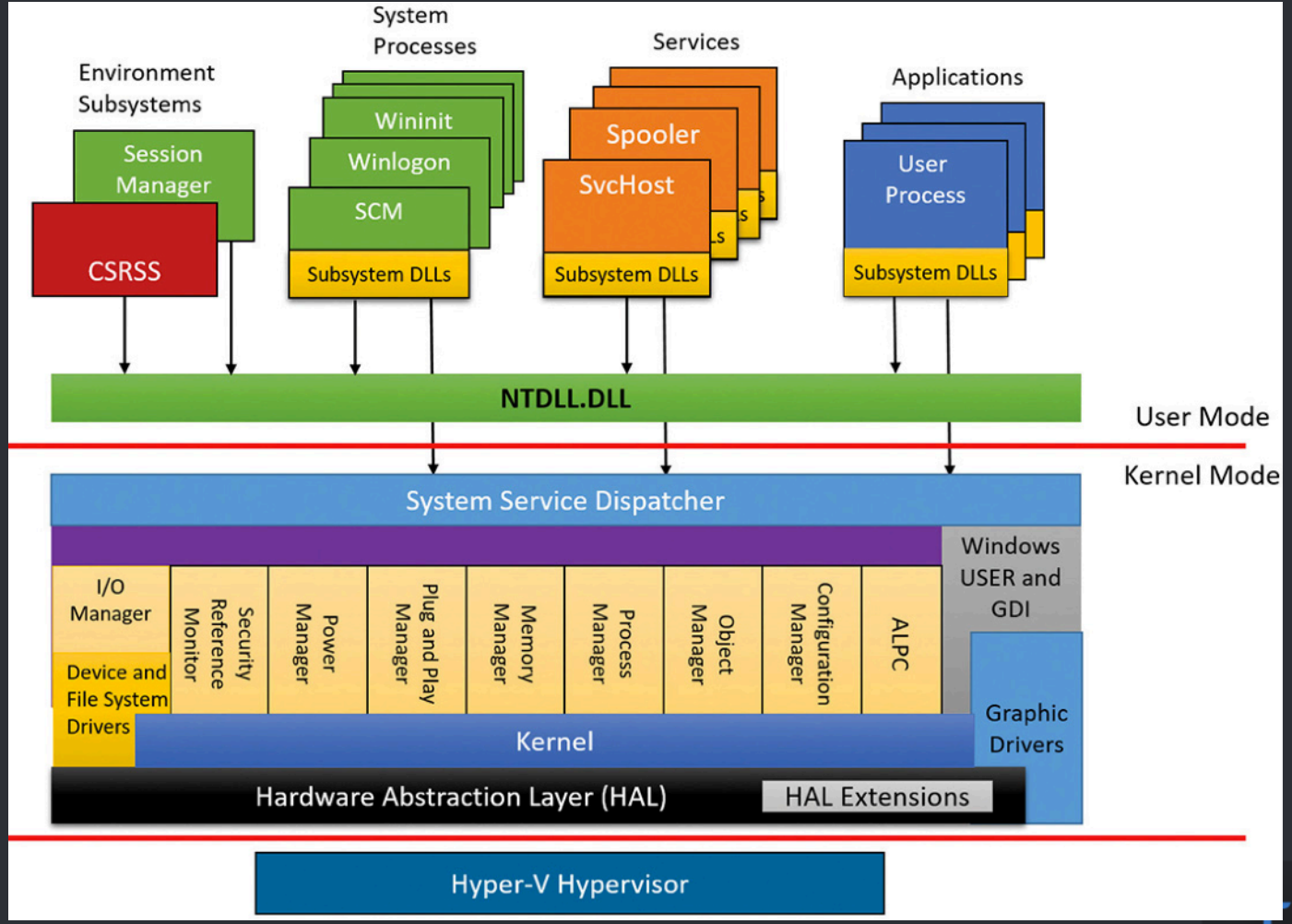
Dump memory

- .writemem *FileName Range*
- .readmem Filename Range

Know your Windows



From: Windows Internals book



From: Windows Internals book

Object enumeration

- !object
- Available object types
  - `.for(r? $t0=0; @$t0 < 40; r? $t0= @$t0+1) { dt nt!_OBJECT_TYPE poi(nt!ObTypeIndexTable + @$t0 * 8) Name }`

Exploring Windows

- _KPCR and _KPCRB
  - PCR (!PCR)
  - dt nt!_KPCR
- _EPROCESS and _KPROCESS
- _OBJECT_HEADER
- Loader
- Objects
- Driver and Device Objects
- IDT, GDT
- SSDT (and shadow)

Loaded modules

- lmv
- lmDm Pattern
- !lmi
- !for_each_module
- !object \Driver
- !handle
- !drvobj
- !devobj
- !devhandles

Processes and threads

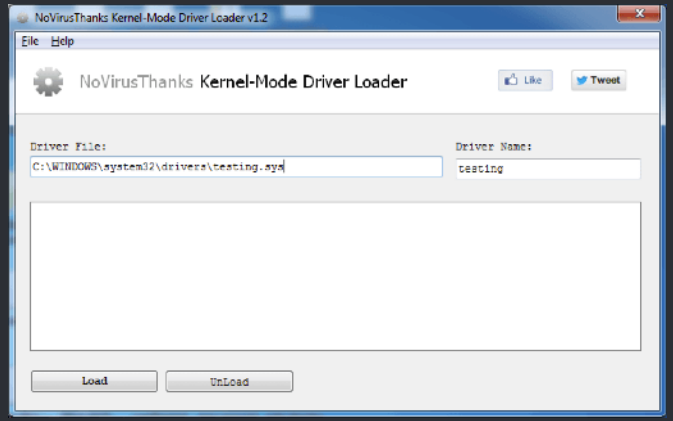
- !process 0 0
- !threads
- .tlist

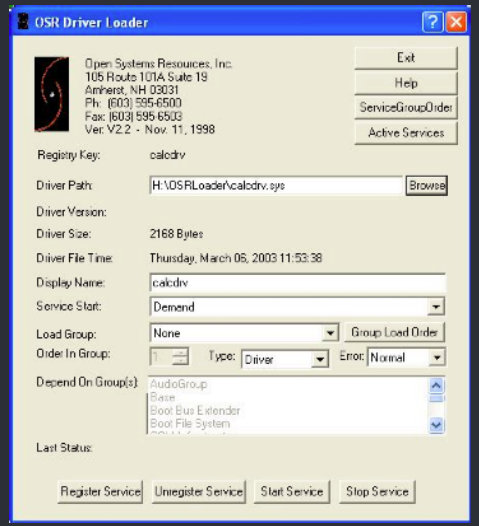
- !for_each_process
- walking csrss.exe handle table
- !peb
- !teb

Expected malware behavior

Loading drivers

- Disable integrity checking
- Enable test signing
- Use one of the utilities
  - OSR Driver loader
  - Novirusthanks





TALOS

Malicious kernel activity

- Hooking code
  - API functions
    - Ntkernel
    - !chkimage (for comparison of symbols)
  - Driver MAJOR function handlers
    - Tcpip.sys
- Hooking data
  - Documented callbacks
  - Undocumented tables
    - Protected so watch for access to cr0

Malicious kernel activity

- Add file systems
- Exploit legacy drivers to disable integrity checks
  - dq ci!g_CiOptions (Windows 8+)
  - dq nt!g_CiEnabled (Windows 7-)

Uroboros/Turla

```
kd> !idt

Dumping IDT: 80b95400

3194895000000030:      82c27ca4 hal!Halp8254ClockInterrupt
3194895000000031:      8486b058 i8042prt!I8042KeyboardInterruptService (KINTERRUPT 8486b000)
3194895000000038:      82c18c6c hal!HalpRtcProfileInterrupt
3194895000000039:      8486bcd8 ACPI!ACPIInterruptServiceRoutine (KINTERRUPT 8486bc80)
319489500000003a:      85afd7d8 ndis!ndisMiniportIsr (KINTERRUPT 85afd780)
319489500000003b:      8486b558 ataport!IdePortInterrupt (KINTERRUPT 8486b500)
319489500000003c:      85afdcd8 i8042prt!I8042MouseInterruptService (KINTERRUPT 85afdc80)
319489500000003e:      8486ba58 ataport!IdePortInterrupt (KINTERRUPT 8486ba00)
319489500000003f:      8486b7d8 ataport!IdePortInterrupt (KINTERRUPT 8486b780)
31948950000000c3:      859e84f0
```

From: GData research

TALOS

Uroboros/Turla

```
kd> u 859e84f0  L0x16
859e84f0 90              nop
859e84f1 90              nop
859e84f2 90              nop
859e84f3 90              nop
859e84f4 90              nop
859e84f5 90              nop
859e84f6 90              nop
859e84f7 90              nop
859e84f8 90              nop
859e84f9 90              nop
859e84fa 90              nop
859e84fb 90              nop
859e84fc 90              nop
859e84fd 90              nop
859e84fe 90              nop
859e84ff 90              nop
859e8500 6a08            push    8
859e8502 6808859e85      push    859E8508h
859e8507 cb              retf
859e8508 fb              sti
859e8509 50              push    eax
859e850a 51              push    ecx
```

From: GData research

Uroboros/Turla

```
kd> ? IoCreateDevice
Evaluate expression: -2103684120 = 829c53e8
kd> u 829c53e8
nt!IoCreateDevice:
829c53e8 6a01            push    1
829c53ea cdc3            int     0C3h
829c53ec ec              in      al,dx
829c53ed 83e4f8          and     esp,0FFFFFFF8h
829c53f0 81ec94000000    sub     esp,94h
829c53f6 a14cda9282      mov     eax,dword ptr [nt!__security_cookie (8292da4c)]
829c53fb 33c4            xor     eax,esp
829c53fd 89842490000000  mov     dword ptr [esp+90h],eax
```

From: GData research

TALOS

Malicious kernel activity - detection

- Enumerate loaded driver objects
  - and associated device objects
- chkimg -d
- Scan for driver major function hooks
- Scan callbacks
- Scan handle tables
- Scan memory for “hidden” modules

Check

- object for scheduled jobs
- kernel threads
- DPCs, APCs

Common (malware) called functions

- CmRegisterCallback - Registry callback for protection of registry values
- PsSetCreateProcessNotifyRoutine - respawning the payload if the payload process is terminated
- PsSetLoadImageNotifyRoutine - to disable User Account Control
- PsSetCreateThreadNotifyRoutine - registry and driver file protection
- ObRegisterCallbacks - to protect the payload from termination
- IoCreateDevice
- IoCreateSymbolic link
- ExAllocatePoolWithTag

Malicious kernel activity - detection

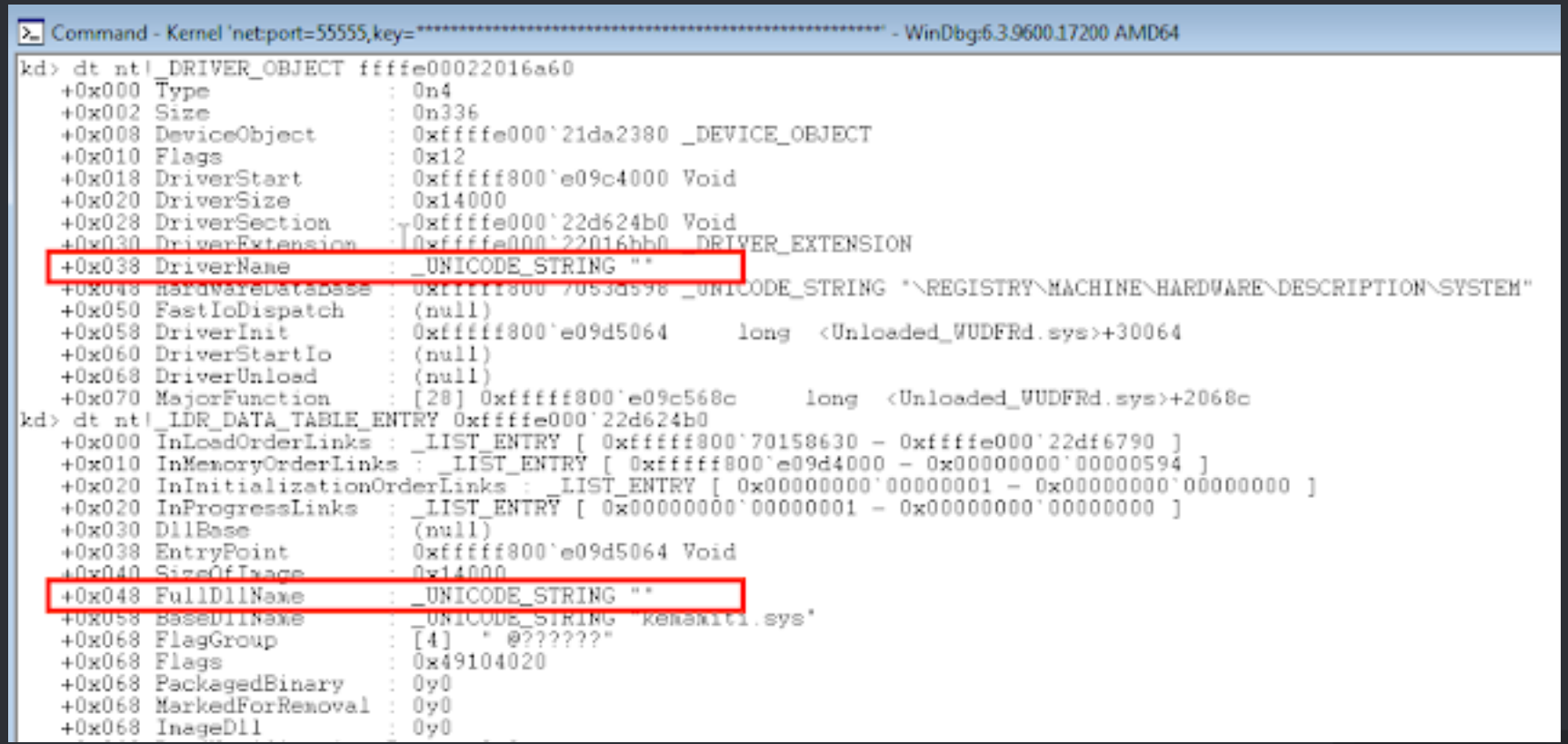
- Enumerate loaded driver objects
  - and associated device objects
- chkimg -d
- Scan for driver major function hooks
- Scan callbacks
- Scan handle tables
- Scan memory for “hidden” modules

Extensions

- swishdbgext (by Matt Suiche)
- wdbgark (by swwwolf)
- dbgkit (by Andrey Bazhan)

- .load
- !extname.help

Zero the driver name



Detection

```
[*] PspCreateProcessNotifyRoutine:
     Procedure: 0xFFFFF8006FF80BAC (nt!ViCreateProcessCallback)
Loading symbols for fffff800`de832000          cng.sys ->    cng.sys
     Procedure: 0xFFFFF800DE835804 (cng!CngCreateProcessNotifyRoutine)
     Procedure: 0xFFFFF800DECF5384 (WdFilter!MpCreateProcessNotifyRoutineEx)
Loading symbols for fffff800`df0ed000       ksecdd.sys ->    ksecdd.sys
     Procedure: 0xFFFFF800DF100000 (ksecdd!KsecCreateProcessNotifyRoutine)
Loading symbols for fffff800`df47e000        tcpip.sys ->    tcpip.sys
     Procedure: 0xFFFFF800DF4FCF20 (tcpip!CreateProcessNotifyRoutineEx)
Loading symbols for fffff800`de530000           CI.dll ->    CI.dll
     Procedure: 0xFFFFF800DE562C70 (CI!I_PEProcessNotify)
Loading symbols for fffff800`e074e000       peauth.sys ->    peauth.sys
*** ERROR: Module load completed but symbols could not be loaded for peauth.sys
     Procedure: 0xFFFFF800E07C8810 (peauth+0x7a810)
     Procedure: 0xFFFFF800E09CA944 (+0x25944)

[*] PspLoadImageNotifyRoutine:
     Procedure: 0xFFFFF800DED12804 (WdFilter!MpLoadImageNotifyRoutine)
     Procedure: 0xFFFFF800E09C78C8 (+0x228c8)

[*] PspCreateThreadNotifyRoutine:
     Procedure: 0xFFFFF800DED121EC (WdFilter!MpCreateThreadNotifyRoutine)
     Procedure: 0xFFFFF800E09C78EC (+0x228ec)

[*] CallbackListHead:
     Procedure: 0xFFFFF800DED0E728 (WdFilter!MpRegCallback)
     Procedure: 0xFFFFF800E09C71CC (+0x221cc)

[*] KeBugCheckCallbackListHead:
Loading symbols for fffff800`df2c4000         ndis.sys ->    ndis.sys
     Procedure: 0xFFFFF800DF3201B8 (ndis!ndisBugcheckHandler)
Loading symbols for fffff800`6fe15000          hal.dll ->    hal.dll
     Procedure: 0xFFFFF8006FE3135C (hal!HalpMiscBugCheckCallback)
```

Scripting

Conditional statements

- .if, .then, .else
- j (ternary) - use with conditional breakpoints
  - bp

Repetition

- .for
- .foreach
- .do
- .while
- .break
- .continue
- .block

Aliases

- aS
- aD
- al
- aS /x myAlias 5 + 1; .block{.echo ${myAlias}}
  - .block idiosyncrasy

Display SSDT - scripting

```
dps nt!KiServiceTable L50

r? @$t3= *(unsigned int *) @@(nt!KiServiceLimit)
r? @$t1= (int *) @@(nt!KiServiceTable)

.for (r? @$t2=0; @$t2 < @$t3 ; r? @$t2=@$t2 + 1) {
      r? @$t4 = @$t1[@$t2] >> 4
     .printf "%y\n", @$t4 +@$t1
}
```

TALOS

Example 1

```
$$ Set t0 to point to the head of the InLoadOrderModuleList of PEB
r? @$t0 = (nt!_LIST_ENTRY *) (&@$peb->Ldr->InLoadOrderModuleList)

$$ Traverse the list by following Flink field and get FullDllName
.for (r? @$t1=@$t0->Flink; @$t0 != @$t1; r? @$t1=@$t1->Flink)
{
    $$ Cast list entry to _LDR_DATA_TABLE_ENTRY (Offset 0)
    $$ to get to the name

    r? @$t2 = (nt!_LDR_DATA_TABLE_ENTRY *) @@(@$t1)
   .printf "%msu\n", @@c++(&@$t2->FullDllName)
}
```

TALOS

Example 2

```
r? @$t0= (nt!_LIST_ENTRY*) @@(nt!PsActiveProcessHead)

.for (r? @$t1= @$t0->Flink;
      (@$t1 != @$t0);
     r? @$t1 = @$t1->Flink)
{
r? @$t2=#CONTAINING_RECORD(@$t1, nt!_EPROCESS, ActiveProcessLinks)
.if (@@(@$t2->BreakOnTermination) == 1)
   {
     as /ma $ProcName @@(@$t2->ImageFileName)
     as /x $CritProc @@(@$t2->BreakOnTermination)
     .block { .echo ${$ProcName} has BreakOnTermination ${$CritProc} }
     ad $ProcName
     ad $CritProc
    }
}
```

Scripting

- Invoking scripts

```
$<Filename
$><Filename
$$<Filename
$$><Filename
$$>a<Filename [arg1 arg2 arg3 ...]
```

Javascript to rescue

Javascript to rescue

- Chakracore engine integrated (EC6 implementation)
- Built on top of debugger object model
- Scripting
- Visualization
- Extending the model

Debugger Object model

- Debugger
- Sessions
- Processes
- Threads
- Stack
- Modules
- Handles
- Local variables
- Settings

Debugger Object model

- dx - new command to investigate
- Utility (send commands to Debugger)

```
host.namespace.Debugger.Utility.Control.ExecuteCommand("u");
```

Debugger Object model accesible from JS

```
// WinDbg JavaScript sample
// Prints Hello World
function initializeScript()
{
    host.diagnostics.debugLog("***> Hello World! \n");
}
```

TALOS

Javascript commands

- .load jsprovider.dll
- .scriptload
- .scriptrun
- .scriptunload
- .scriptlist
- .scriptproviders

Javascript entry points

- root
- invokeScript()
- initializeScript()
- uninitializeScript()

64 bit problems

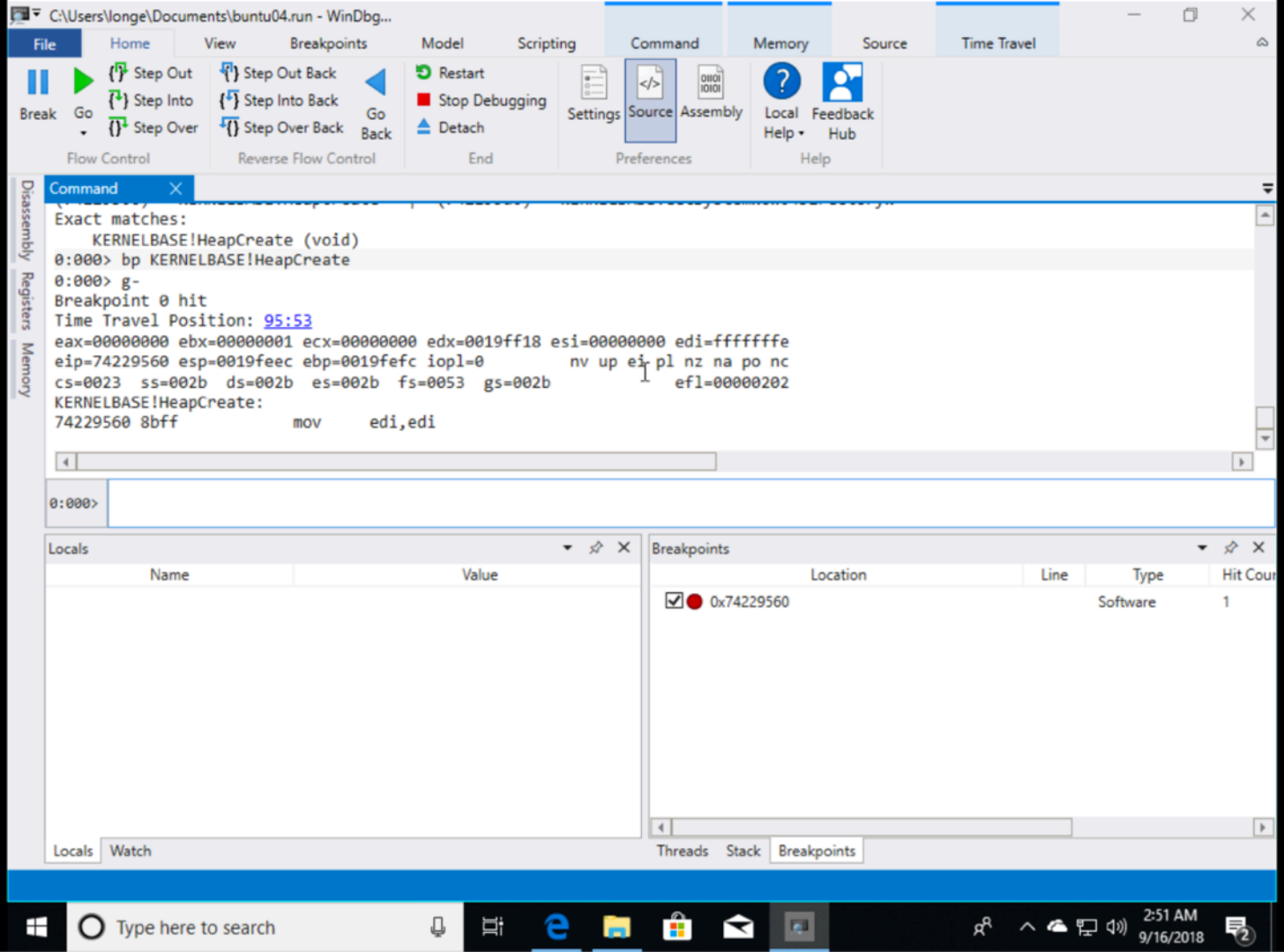
- Javascript integers only 53 bit
- Special data class Int64 and the methods

Linq

- Language Integrated Query
- `dx @$curprocess.Modules.Select(m => m.Name).Where(n => n.Contains("maldll"))`
- `dx @$currsession.TTD.Calls().Count()`

Time travel debugging

- Record a trace
- move forwards and backwards “in time”
- Set breakpoint on an API call and go backwards
- p-
- g-
- t-


C:\Users\longe\Documents\buntu04.run - WinDbg...
File
Home
View
Breakpoints
Model
Scripting
Command
Memory
Source
Time Travel
Break
Go
Step Out
Step Into
Step Over
Step Out Back
Step Into Back
Step Over Back
Go Back
Restart
Stop Debugging
Detach
Settings
Source
Assembly
Local Help
Feedback Hub
Flow Control
Reverse Flow Control
End
Preferences
Help
Disassembly
Registers
Memory
Command
Exact matches:
KERNELBASE!HeapCreate (void)
0:000> bp KERNELBASE!HeapCreate
0:000> g-
Breakpoint 0 hit
Time Travel Position: 95:53
eax=00000000 ebx=00000001 ecx=00000000 edx=0019ff18 esi=00000000 edi=fffffffe
eip=74229560 esp=0019feec ebp=0019fefc iopl=0 nv up ei pl nz na po nc
cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000202
KERNELBASE!HeapCreate:
74229560 8bff mov edi,edi
0:000>
Locals
Name
Value
Locals
Watch
Breakpoints
Location
Line
Type
Hit Cou
0x74229560
Software
1
Threads
Stack
Breakpoints
Type here to search
2:51 AM
9/16/2018

Extensions

Loading and Checking Extensions

- .load
- .loadby
- .chain
- version

Extensions

- Jsprovider
- swishdbgext
- wdbgark
- dbgkit
- mex
- sos
- Pykd

Scripting - pyKD

- Python extension to make scripting easier

  !py pykdexample.py

```
#!/usr/bin/env python
from pykd import *

zwcreateapis=[]
zwcreateapis= dbgCommand("x nt!ZwCreate*")

for api in zwcreateapis.split("\n"):
    print api.split(" ")[1] #print name
```

Relax and breathe!

TaLOS

Enabling the Good Guys

Spreading security news, updates, and other information to the public

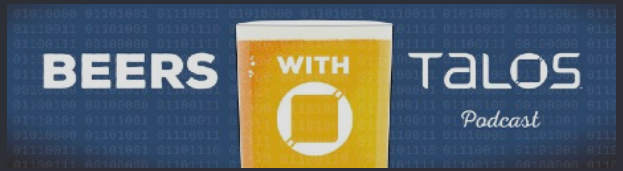
White papers, articles, & other information
talosintelligence.com

ThreatSource Newsletter
cs.co/TalosUpdate

Talos Blog
blog.talosintelligence.com

Social Media Posts
Facebook: TalosGroupatCisco
Twitter: @talossecurity

Instructional Videos
cs.co/talostube



Download on iTunes

GET IT ON Google Play

TALOS

References - setup

- https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/
- https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/getting-set-up-for-debugging
- https://www.contextis.com/blog/introduction-debugging-windows-kernel-windbg
- https://reverseengineering.stackexchange.com/questions/2297/windows-kernel-debugging-on-mac-host-using-vmware-fusion#2298
- https://communities.vmware.com/docs/DOC-15691 - vm to vm over a virtual serial port VMWare Windows

References - malware analysis

- http://blog.talosintelligence.com/2017/08/windbg-and-javascript-analysis.html
- http://blog.talosintelligence.com/2017/07/unravelling-net-with-help-of-windbg.html
- https://www.gdatasoftware.com/blog/2014/06/23953-analysis-of-uroburos-using-windbg
- http://www.sekoia.fr/blog/wp-content/uploads/2016/10/Rootkit-analysis-Use-case-on-HIDEDRV-v1.6.pdf
- https://www.youtube.com/watch?v=l2ZSG_96PoM
- https://www.offensive-security.com/vulndev/fldbg-a-pykd-script-to-debug-flashplayer/

TALOS

References - Javascript and object model

- https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/dx--display-visualizer-variables-
- https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/using-linq-with-the-debugger-objects
- https://doar-e.github.io/blog/2017/12/01/debugger-data-model/

References - others

- http://www.zachburlingame.com/2011/12/customizing-your-windbg-workspace-and-color-scheme/ - Workspace setup
- https://github.com/vagnerpilar/windbgtree - cmdtree
- https://github.com/vallejocc/Reverse-Engineering-Arsenal/tree/master/WinDbg - WinDbg scripting 1
- https://archive.codeplex.com/?p=kdar - WinDbg scripting 2 - Archive available
- https://githomelab.ru/pykd/pykd/wikis/User%20Manual%20rus - PyKD manual - Russian only, translates OK
- http://windbg.info/download/doc/pdf/WinDbg_cmds.pdf - WinDbg commands cheatsheet
- https://www.youtube.com/watch?v=vz15OqiYYXo&feature=share - Windows Internals by Alex Sotirov
- http://terminus.rewolf.pl/terminus/ - Project Terminus Undocumented Structures Diff

TALOS

References - driver loading tools

- https://www.osronline.com/article.cfm?article=157
- http://www.novirusthanks.org/products/kernel-mode-driver-loader/
- https://github.com/maldevel/driver-loader

References - extensions

- https://www.microsoft.com/en-us/download/details.aspx?id=53304 - Mex
- https://github.com/comaeio/SwishDbgExt
- https://github.com/swwwolf/wdbgark
- https://githomelab.ru/pykd/pykd/wikis/Pykd%20bootstrapper - PyKD
- https://github.com/corelan/windbglib - windbglib and mona.py
- https://github.com/pstolarz/dumpext - extension for dumping PE from memory
- http://www.andreybazhan.com/dbgkit.html - Dbgkit

References - books

- Practical Reverse Engineering: x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation (Chapters 3 and 4)
- Practical Malware Analysis: A Hands-On Guide to Dissecting Malicious Software (Chapter 10)
- Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code (Chapter 14)
- The Art Of Memory Forensics – Detecting Malware and Threats in Windows, Linux and Mac Memory
- Rootkit Arsenal
- Advanced Windows Debugging
- Windows Internals
- Windows NT Device Driver Development

References - videos

- https://www.youtube.com/playlist?list=PLhx7-txsG6t6n_E2LgDGqgvJtCHPL7UFu - WinDbg tutorials by TheSourceLens
- https://www.youtube.com/watch?v=s5gOW-N9AAo&list=PLb07KvumDAnD39kssVz7DgmvNH5j89k3b Hacking Livestream #28: Windows Kernel Debugging Part I
- https://channel9.msdn.com/Shows/Defrag-Tools/Defrag-Tools-170-Debugger-JavaScript-Scripting - WinDbg JavaScript scripting
- https://channel9.msdn.com/Shows/Defrag-Tools/Defrag-Tools-138-Debugging-dx-Command-Part-1 - Dx command part 1 (and then 2)
- https://channel9.msdn.com/Shows/Defrag-Tools/Defrag-Tools-169-Debugging-Tools-for-Windows-Team - for Debugger object model
- https://www.youtube.com/watch?v=l1YJTg_A914 - Time Travel Debugging

TALOS

talosintel.com
@talossecurity
@vanjasvajcer

isvajcer@cisco.com