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 (1394)
  - USB
  - Network (TCP/IP)
Live debugging setup - VM to VM

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

- .cmdtree - useful for learning and remembering commands

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

```bash
!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
Object enumeration

• !object
• Available object types
  • .for($t0=0; $t0 < 40; $t0= @$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
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

From: GData research

```plaintext
kd> !idt

Dumping IDT: 80b95400

<table>
<thead>
<tr>
<th>Address</th>
<th>Microsoft Windows Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000030:</td>
<td><code>hal!Halp8254ClockInterrupt</code></td>
</tr>
<tr>
<td>0000000000031:</td>
<td><code>i8042prt!I8042KeyboardInterruptService (KINTERRUPT 8486b000)</code></td>
</tr>
<tr>
<td>0000000000038:</td>
<td><code>hal!HalpRtcProfileInterrupt</code></td>
</tr>
<tr>
<td>0000000000039:</td>
<td><code>ACPI!ACPIInterruptServiceRoutine (KINTERRUPT 8486bc80)</code></td>
</tr>
<tr>
<td>000000000003a:</td>
<td><code>ndis!ndisMiniportIsr (KINTERRUPT 85afd780)</code></td>
</tr>
<tr>
<td>000000000003b:</td>
<td><code>ataport!IdePortInterrupt (KINTERRUPT 8486b500)</code></td>
</tr>
<tr>
<td>000000000003c:</td>
<td><code>i8042prt!I8042MouseInterruptService (KINTERRUPT 85afd80)</code></td>
</tr>
<tr>
<td>000000000003d:</td>
<td><code>ataport!IdePortInterrupt (KINTERRUPT 8486ba00)</code></td>
</tr>
<tr>
<td>000000000003e:</td>
<td><code>ataport!IdePortInterrupt (KINTERRUPT 8486b780)</code></td>
</tr>
<tr>
<td>000000000003f:</td>
<td><code>859e84f0</code></td>
</tr>
</tbody>
</table>
```

From: GData research
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>859e84f0</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e84f1</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e84f2</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e84f3</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e84f4</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e84f5</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e84f6</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e84f7</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e84f8</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e84f9</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e84fa</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e84fb</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e84fc</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e84fd</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e84fe</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e84ff</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>859e8500</td>
<td>6a08</td>
<td>push 8</td>
</tr>
<tr>
<td>859e8502</td>
<td>6808859e85</td>
<td>push 859e8508h</td>
</tr>
<tr>
<td>859e8507</td>
<td>cb</td>
<td>retf</td>
</tr>
<tr>
<td>859e8508</td>
<td>fb</td>
<td>sti</td>
</tr>
<tr>
<td>859e8509</td>
<td>50</td>
<td>push eax</td>
</tr>
<tr>
<td>859e850a</td>
<td>51</td>
<td>push ecx</td>
</tr>
</tbody>
</table>

From: GData research
From: GData research
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 fffff8000de832000 cng.sys -> cng.sys
  Procedure: 0xFFFFF8006D35804 (cng!CngCreateProcessNotifyRoutine)
  Procedure: 0xFFFFF8006D65384 (WdFilter!MpCreateProcessNotifyRoutineEx)
  Loading symbols for fffff8000d0ed000 ksecdd.sys -> ksecdd.sys
  Procedure: 0xFFFFF8006DCF000 (ksecdd!KsecCreateProcessNotifyRoutine)
  Loading symbols for fffff8000d47e000 tcpip.sys -> tcpip.sys
  Procedure: 0xFFFFF8006DF4FC20 (tcpip!CreateProcessNotifyRoutineEx)
  Loading symbols for fffff8000d530000 CI.dll -> CI.dll
  Procedure: 0xFFFFF8006E562C70 (CI!I_PEGProcessNotify)
  Loading symbols for fffff8000e074e000 peauth.sys -> peauth.sys
  *** ERROR: Module load completed but symbols could not be loaded for peauth.sys
     Procedure: 0xFFFFF8006E07C810 (peauth+0x7a810)
     Procedure: 0xFFFFF800E09CA944 (+0x25944)

[*] PspLoadImageNotifyRoutine:
  Procedure: 0xFFFFF8006E12804 (WdFilter!MpLoadImageNotifyRoutine)
  Procedure: 0xFFFFF800E09C78C8 (+0x228c8)

[*] PspCreateThreadNotifyRoutine:
  Procedure: 0xFFFFF8006E121E8 (WdFilter!MpCreateThreadNotifyRoutine)
  Procedure: 0xFFFFF800E09C78EC (+0x228ec)

[*] CallbackListHead:
  Procedure: 0xFFFFF8006E12728 (WdFilter!MpRegCallback)
  Procedure: 0xFFFFF800E09C71CC (+0x221cc)

[*] KeBugCheckCallbackListHead:
  Loading symbols for fffff8000df2c4000 ndis.sys -> ndis.sys
  Procedure: 0xFFFFF8006FE3201B8 (ndis!ndisBugcheckHandler)
  Loading symbols for fffff8000d0e45000 hal.dll -> hal.dll
  Procedure: 0xFFFFF8006FE3135C (hal!HalMiscBugCheckCallback)
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
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
}
Example 1

```c
// Set t0 to point to the head of the InLoadOrderModuleList of PEB
@$t0 = (nt!_LIST_ENTRY *) (&@$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
    @$t2 = (nt!_LDR_DATA_TABLE_ENTRY *) @@(@$t1)
    printf "%msu\n", @@c++(&@$t2->FullDllName)
}
```
Example 2

```c
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
$$<Filename
$$>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)

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

- .load jsprovider.dll
- .scriptload
- .scriptrun
- .scriptunload
- .scriptlist
- .scriptproviders
Javascript entry points

- root
- invokeScript()
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-
Time travel debugging

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 es=00000000 edi=fffffffff
esi=74229560 esp=0019feec ebp=0019fefe iopl=0 ncpu up ei pl nz na po nc
psw=00 cs=0023 ss=002b ds=002b es=002b fs=0053 gs=002b
efl=000000202
KERNELBASE!HeapCreate: 74229560 8bff mov edi,edi

Locals
Name        Value

Breakpoints
Location    Line Type Hit Count
0x74229560

Locals Watch
Extensions
Loading and Checking Extensions

- .load
- .loadby
- .chain
- version
Extensions

- Jsprovider
- swishdbgext
- wdbgark
- dbgkit
- mex
- sos
- Pykd
• Python extension to make scripting easier

!py pykdexample.py

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

dzwcreateapis=dbgCommand("x nt!ZwCreate*")

for api in zwcreateapis.split("\n"):
    print api.split(" ")[1] #print name
Relax and breathe!
Enabling the Good Guys

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

ThreatSource Newsletter
cs.co/TalosUpdate

Social Media Posts
Facebook: TalosGroupatCisco
Twitter: @talossecurity

White papers, articles, & other information
talosintelligence.com

Talos Blog
blog.talosintelligence.com

Instructional Videos
cs.co/talostube

BEERS WITH TALOS Podcast
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/)
- [https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/getting-set-up-for-debugging](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://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://reverseengineering.stackexchange.com/questions/2297/windows-kernel-debugging-on-mac-host-using-vmware-fusion#2298)
- [https://communities.vmware.com/docs/DOC-15691](https://communities.vmware.com/docs/DOC-15691) - vm to vm over a virtual serial port VMWare Windows
References - malware analysis

- https://www.youtube.com/watch?v=lZSG_96PoM
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/dx--display-visualizer-variables)
- [https://doar-e.github.io/blog/2017/12/01/debugger-data-model/](https://doar-e.github.io/blog/2017/12/01/debugger-data-model/)
References - others

- https://github.com/vagnerpilar/windbgtree - cmdtree
- https://github.com/vallejocc/Reverse-Engineering-Arsenal/tree/master/Windbg - WinDbg scripting 1
- https://www.youtube.com/watch?v=vz15OqiYYXo&feature=share - Windows Internals by Alex Sotirov
- http://terminus.rewolf.pl/terminus/ - Project Terminus Undocumented Structures Diff
References - driver loading tools

- [https://www.osronline.com/article.cfm?article=157](https://www.osronline.com/article.cfm?article=157)
- [https://github.com/maldevel/driver-loader](https://github.com/maldevel/driver-loader)
References - extensions

- 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_F2LgDGqevJtCHPl7UFu](https://www.youtube.com/playlist?list=PLhx7-txsG6t6n_F2LgDGqevJtCHPl7UFu) - WinDbg tutorials by TheSourceLens
- [https://www.youtube.com/watch?v=s5gOW-N9AAo&list=PLb07KvumDAnD39kssVz7DgmvNH5j89k3b](https://www.youtube.com/watch?v=s5gOW-N9AAo&list=PLb07KvumDAnD39kssVz7DgmvNH5j89k3b) - Hacking Livestream #28: Windows Kernel Debugging Part I
- [https://www.youtube.com/watch?v=l1YJTg_A914](https://www.youtube.com/watch?v=l1YJTg_A914) - Time Travel Debugging
talosintel.com
@talosssecurity
@vanjasvajcer

isvajcer@cisco.com