Characterization Of The Windows Kernel Version Variability For Accurate Memory Analysis

By

Michael Cohen

Presented At

The Digital Forensic Research Conference

DFRWS 2015 EU Dublin, Ireland (Mar 23rd- 26th)

Characterization of the Windows Kernel version variability for accurate Memory analysis.

Michael Cohen
scudette@google.com

DFRWS 2015 Dublin.

Memory Analysis overview

How do we analyse memory?

We need to emulate the way code looks at memory.

Data Structures

```c
typedef unsigned char uchar;

enum {
  OPT1,
  OPT2
} options;

struct foobar {
    enum options flags;
    short int bar;
    uchar *foo;
}
```

It is generally not possible to predict the memory layout of a C struct without knowing external factors:

- Alignment
- Endianess
- Bit size (64/32 bit)
- Compiler
- Optimizations etc

Unless packed structs.

Example memory analysis technique

- Listing processes
  - Find the global kernel symbol "PsActiveProcessHead"
  - Follow the linked list _EPROCESS.ActiveProcessLinks to find all _EPROCESS structs.
  - Print _EPROCESS.ImageFileName

Kernel Constant

Struct Offset

Struct Offset

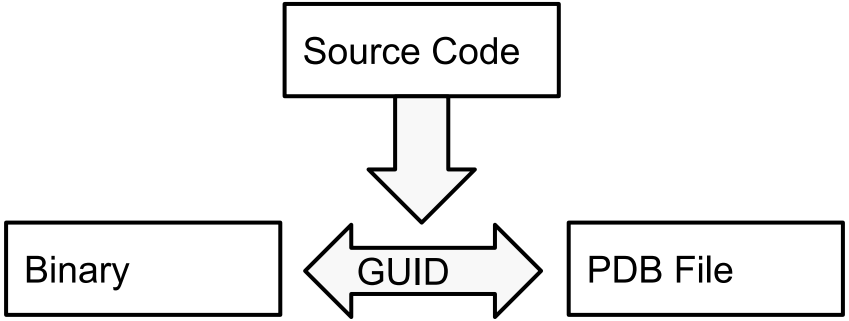
Data Structures

```
typedef unsigned char uchar;

enum {
  OPT1,
  OPT2
} options;

struct foobar {
    enum options flags;
    short int bar;
    uchar *foo;
}
```

Debugging symbols contain the exact layout of all data structures.

Can use them to get struct offset AND kernel global constants.

What do PDB files look like?



1. Each time the binary is built, a GUID is generated.
2. The debugging symbols are stored in a PDB file.
3. The executable is shipped.
4. PDB files can be made available publicly on a **symbol server**.

Historical perspective

- Older tools have profiles embedded inside the tool.
  - Profile is pre-generated from an exemplar of an OS version released.
    - e.g. Win7SP1x64
  - Profile is embedded inside the tool
  - We assume profile is applicable to all releases of this version.
  - Profile only contains structs - no use of global offsets from PDB file.
  - Global offsets are deduced by scanning.

Can we always guess kernel globals?

ADD -- Complicating Memory Forensics Through Memory Disarray

Jake Williams and Alissa Torres

In this presentation, we'll present ADD (attention deficit disorder), a tool that litters Windows physical memory with (configurable amounts and types of) garbage to disrupt memory forensics. Memory forensics has become so mainstream that it's catching too many malware authors during routine investigations (making Jake a sad panda). If memory forensics were much harder to perform, then attackers would retain an upper hand. ADD increases the cost of memory forensics by allocating new structures in memory that serve only to disrupt an investigation.

We'll present some basic memory forensics techniques (just to set the stage for those who aren't familiar with the concepts). We'll explain how volatility, a core memory forensics tool, actually performs its analysis. In particular, we'll show how it locates hidden processes, drivers, and modules.

Next, we'll show how running ADD on a machine under investigation completely changes the memory forensics landscape. We'll show how an investigator must weed through astounding numbers of false positives before identifying the investigation targets.

Finally, Alissa will show how all is not lost. Even though ADD may confuse junior analysts, she'll show the invariants in memory that analysts should always be able to come back to complete their forensic analysis.

Jake is the Chief Scientist at CSRgroup where he does lots of offensive and defensive research. He is also a SANS instructor and member of the DFIR author team. Occasionally, CSRgroup still lets Jake do penetration tests (where he feels like a kid in a candy store).

Alissa is a digital forensics examiner and incident response consultant for Sibertor Forensics. Also a SANS Instructor, she teaches hundreds of security professionals a year how to find evil in the form of trace artifacts and hidden processes.

Can we do better?

- Rekall chooses to rely on constants obtained from debugging symbols.
  - Pros
    - Better coverage of symbols, especially ones that are not exported.
  - Cons
    - We need to wait for Microsoft to make a pdb file available for us to use.

Lets evaluate this approach.

- Basic assumption in Volatility:
  - Struct layout does not change between major and minor versions.
    - An exemplar from a particular version will apply to all kernels from that version.
  - Kernel global symbols vary too much between major and minor versions to hard code
    - Therefore we need to scan for them.

This repository Search or type a command Explore Gist Blog Help scudette

google / rekall-profiles

Watch 3 Star 1 Fork 1

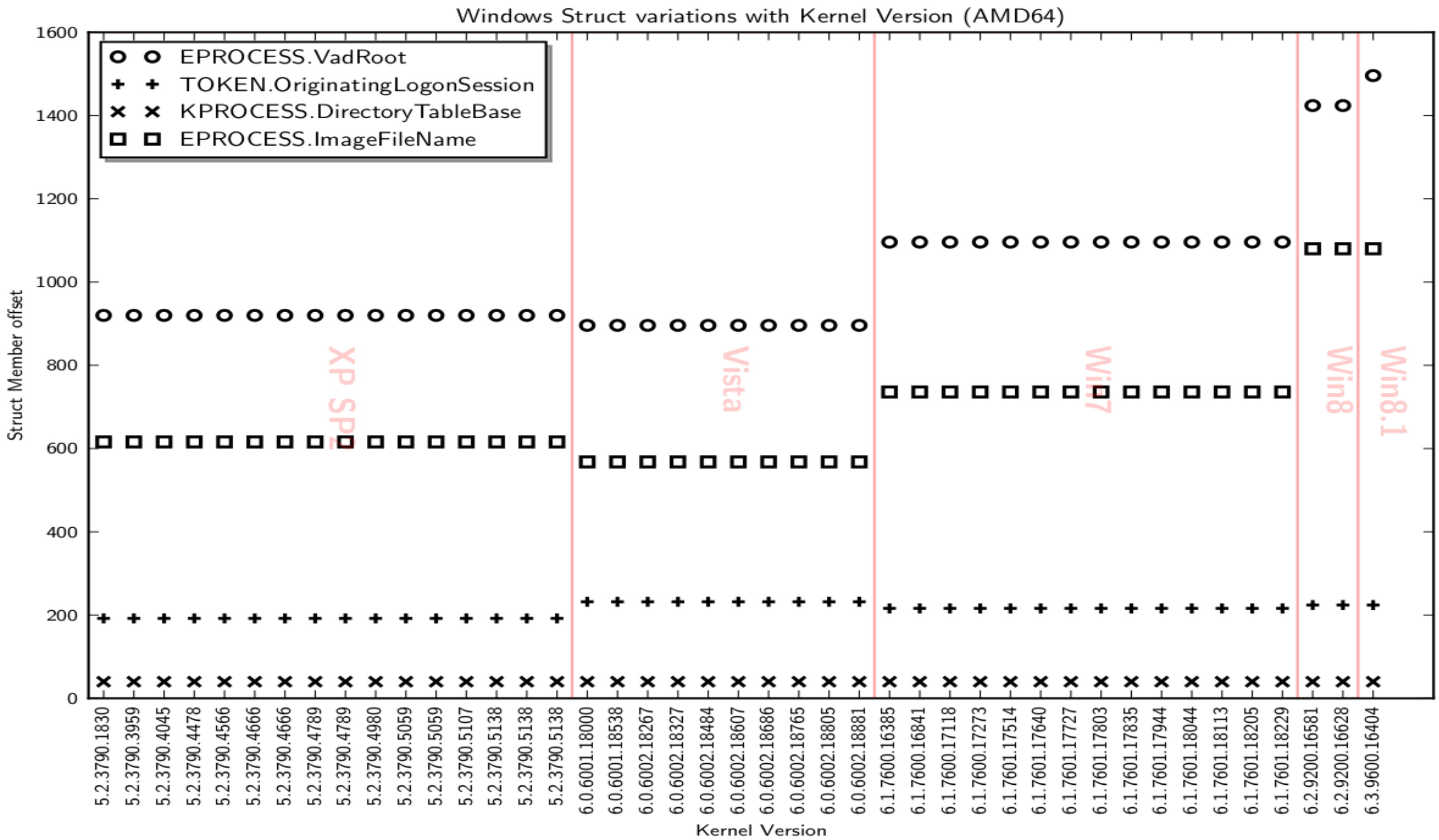
branch: master rekall-profiles / v1.0 / nt / GUID / +

Updated repo index, removed obsolete html files.

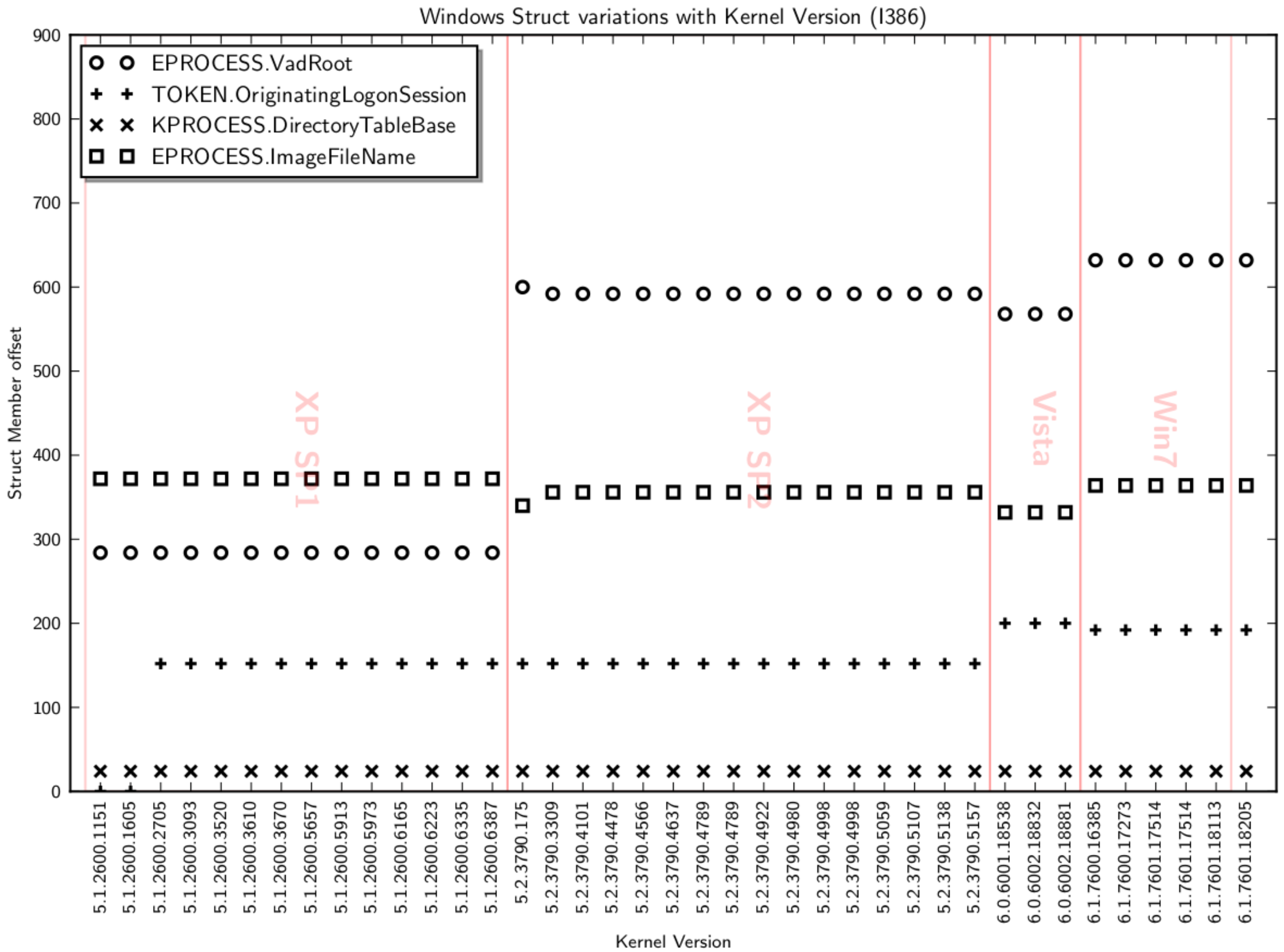
the80srobot authored on Jun 6 latest commit 0142f0aad1

..

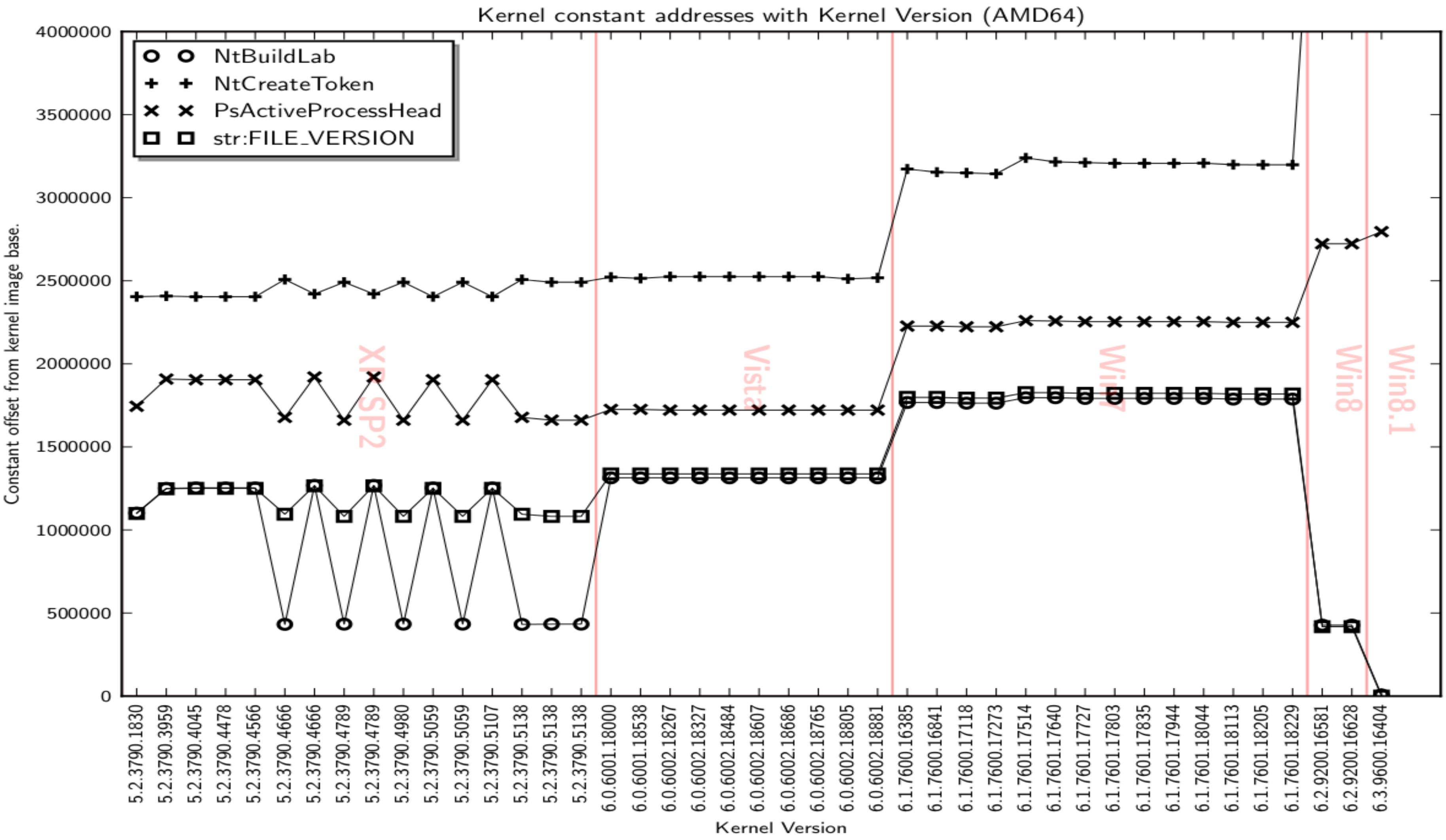
00625D7D36754CBEBA4533BA9A0F3FE22.gz	Initial push of profiles into the profile repository.	3 months ago
0100FCDAFD4049B8B06005EC07705A1F2.gz	Initial push of profiles into the profile repository.	3 months ago
01DDCBD82AEB46BEAFCDC6A409E3B1D31…	Initial push of profiles into the profile repository.	3 months ago
01DF28C698D84DEBB1A74254C3AF800E2.gz	Initial push of profiles into the profile repository.	3 months ago
0318508323324 9D9BB747EA777B80C982.gz	Initial push of profiles into the profile repository.	3 months ago
04FB9A156FF44ECCA6EBCAE9617D8DB73.gz	Initial push of profiles into the profile repository.	3 months ago
05A6F49C5DD848FF983459421A78F1232.gz	Initial push of profiles into the profile repository.	3 months ago
06472CCD0ECF43B58D676891C6745DAC2.gz	Initial push of profiles into the profile repository.	3 months ago
0887873AD7FC4115AC9258B9871F81341.gz	Initial push of profiles into the profile repository.	3 months ago
08F4D00C3B5A4B34B2D3AE8402F927802.gz	Initial push of profiles into the profile repository.	3 months ago



Windows Struct variations with Kernel Version (AMD64)
EPROCESS.VadRoot
TOKEN.OriginatingLogonSession
KPROCESS.DirectoryTableBase
EPROCESS.ImageFileName
Struct Member offset
Kernel Version
XP SP2
Vista
Win7
Win8
Win8.1
0
200
400
600
800
1000
1200
1400
1600
5.2.3790.1830
5.2.3790.3959
5.2.3790.4045
5.2.3790.4478
5.2.3790.4566
5.2.3790.4666
5.2.3790.4666
5.2.3790.4789
5.2.3790.4789
5.2.3790.4980
5.2.3790.5059
5.2.3790.5059
5.2.3790.5107
5.2.3790.5138
5.2.3790.5138
5.2.3790.5138
6.0.6001.18000
6.0.6001.18538
6.0.6002.18267
6.0.6002.18327
6.0.6002.18484
6.0.6002.18607
6.0.6002.18686
6.0.6002.18765
6.0.6002.18805
6.0.6002.18881
6.1.7600.16385
6.1.7600.16841
6.1.7600.17118
6.1.7600.17273
6.1.7601.17514
6.1.7601.17640
6.1.7601.17727
6.1.7601.17803
6.1.7601.17835
6.1.7601.17944
6.1.7601.18044
6.1.7601.18113
6.1.7601.18205
6.1.7601.18229
6.2.9200.16581
6.2.9200.16628
6.3.9600.16404



Google | Rekall



Kernel constant addresses with Kernel Version (AMD64)
NtBuildLab
NtCreateToken
PsActiveProcessHead
str:FILE_VERSION
Constant offset from kernel image base.
Kernel Version
XP SP2
Vista
Win7
Win8
Win8.1
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
5.2.3790.1830
5.2.3790.3959
5.2.3790.4045
5.2.3790.4478
5.2.3790.4566
5.2.3790.4666
5.2.3790.4666
5.2.3790.4789
5.2.3790.4789
5.2.3790.4980
5.2.3790.5059
5.2.3790.5059
5.2.3790.5107
5.2.3790.5138
5.2.3790.5138
5.2.3790.5138
6.0.6001.18000
6.0.6001.18538
6.0.6002.18267
6.0.6002.18327
6.0.6002.18484
6.0.6002.18607
6.0.6002.18686
6.0.6002.18765
6.0.6002.18805
6.0.6002.18881
6.1.7600.16385
6.1.7600.16841
6.1.7600.17118
6.1.7600.17273
6.1.7601.17514
6.1.7601.17640
6.1.7601.17727
6.1.7601.17803
6.1.7601.17835
6.1.7601.17944
6.1.7601.18044
6.1.7601.18113
6.1.7601.18205
6.1.7601.18229
6.2.9200.16581
6.2.9200.16628
6.3.9600.16404

Result

- Assumption is mostly validated for the kernel
  - Struct offsets do not vary per version.
  - Kernel constant very wildly.
- So the Volatility approach should work in most cases!

volatility

An advanced memory forensics framework

Search projects

Project Home | Export to GitHub

New issue | Search | Open issues | for | Search | Advanced search | Search tips | Subscriptions

Issue 174: [profile offsets per build/revision number] was: Process Owner Info not found for TCP_ENDPOINT scan output on Vista (netscan)

8 people starred this issue and may be notified of changes. Back to list

Status: Duplicate
Merged: issue 521
Owner: michael.hale@gmail.com
Closed: Feb 18
Cc: michael.hale@gmail.com, mike.auty@gmail.com, jamie.l...@gmail.com, labaru...@gmail.com, atc...@gmail.com

Type-Defect
Priority-Low
Milestone-3.0.x

Add a comment below

Reported by welcome....@gmail.com, Jan 3, 2012

```
What steps will reproduce the problem?
1. Running netscan plugin on Vista SP2 dump
2.
3.

What is the expected output? What do you see instead?
PID, Process name is not found TCPEndPoint scan output. However that information is present in Listener, UDP scan output.

0xcf7c0de8 TCPv4    0.0.0.0:61618              0.0.0.0:0           LISTENING        2016     spoolsv.exe
0x1cedd88  TCPv4    0.0.0.0:57883              0.0.0.0:443         CLOSED           -------- -------------
0x14e85e08 TCPv4    10.130.179.251:64286       10.177.226.18:1533  ESTABLISHED      -------- -------------

What version of the product are you using? On what operating system?
using volatility 2 standalone python precompiled version on vista sp2.

Please provide any additional information below.
```

Labels: -Priority-Medium Priority-Low

Project Member #17 michael.hale@gmail.com May 18, 2012

```
Here's a little more info.

6.0.6002.18272
Microsoft Windows Server 2008 Standard
6.0.6002 Service Pack 2 Build 6002
push    dword ptr [edi+164h]
call      ds:__imp__PsGetProcessId@4

6.0.6002.18519
Microsoft Windows VistaT Enterprise
6.0.6002 Service Pack 2 Build 6002
push    dword ptr [edi+164h]
call      ds:__imp__PsGetProcessId@4

6.0.6002.18005
Microsoft Windows Server 2008 Standard (also seen Microsoft Windows VistaT Ultimate)
6.0.6002 Service Pack 2 Build 6002
push    dword ptr [edi+160h]
call      ds:__imp__PsGetProcessId@4
```

As you can see, regardless of whether the OS identifies itself as "Server 2008" or "Vista" if the revision number > 18005 then the offset is 0x164. If the revision number <= 18005 then the offset is 0x160. I don't have tcpip.sys binaries for every revision, so I'm not sure if 18005 is exactly where the line is drawn.

Unfortunately until we can choose vtype offsets based on revision number (in addition to the major, minor, build, and memory model which is already possible), then there's not a good way to handle this. I'm not sure we should close this issue since currently we won't print process information for VistaSP2x86/Win2008SP2x86 whose revision numbers are > 18005. However, we might have to defer to fixing it later. In the meantime, manish, you'll have to use a version of volatility where you can change the vtype offset (and if needed build your own exe from it).

Project Member #18 michael.hale@gmail.com Mar 7, 2014

(No comment was entered for this change.)

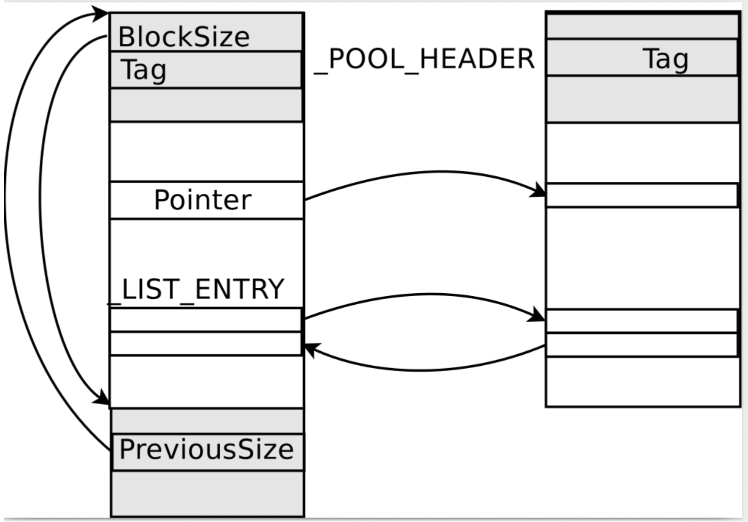
Summary: [profile offsets per build/revision number] was: Process Owner Info not found for TCP_ENDPOINT scan output on Vista (netscan) *(was: Process Owner Info not found for TCP_ENDPOINT scan output on Vista (netscan))*
Cc: -scude...@gmail.com

There are some problems

- Sometimes even struct layout changes within the major/minor version release cycle.
- Mr. Hale is an expert reverse engineer
  - He can figure out the correct struct layout by looking at the disassembly.
  - He knows how to change the program to account for this version.
  - We don't really know how to reproduce:
    - What function was reversed?
    - Which instructions should we look at?

We need automated reversing

- First approach:
  - Look at the data and surrounding context



```
[1] win7.elf.E01 07:20:02> dump "*win32k!grpWinStaList"
-------------------------> dump("*win32k!grpWinStaList")
    Offset                                       Data
-------------- ------------------------------------------------------------------ ----------
0xfa800225ef60 00 00 00 00 00 00 00 00 70 1a 85 01 80 fa ff ff  ........p.......
0xfa800225ef70 30 4b 2c 02 80 fa ff ff 40 f3 3a 00 60 f9 ff ff  0K,.....@.:.`...
0xfa800225ef80 00 00 00 00 00 00 00 00 f0 c9 15 c0 00 f9 ff ff  ................
0xfa800225ef90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0xfa800225efa0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0xfa800225efb0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0xfa800225efc0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0xfa800225efd0 00 00 00 00 00 00 00 00 20 10 87 02 a0 f8 ff ff  ................
0xfa800225efe0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0xfa800225eff0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0xfa800225f000 00 00 58 02 6d 6f 6e 69 eb 25 8b 02 00 00 00 00  ..X.moni.%......
0xfa800225f010 10 f0 25 02 80 fa ff ff 70 7e 92 00 80 fa ff ff  ..%.....p~......
0xfa800225f020 b0 c4 f0 00 80 f8 ff ff 02 10 40 03 03 00 00 00  ..........@.....
0xfa800225f030 70 7e 92 00 80 fa ff ff 1a 0c 01 00 00 00 00 00  p~..............
```

```
[1] win7.elf.E01 07:29:07> analyze_struct "*win32k!grpWinStaList"
--------------------------> analyze_struct("*win32k!grpWinStaList")
0xfa800225ef60 is inside pool allocation with tag 'Win\xe4' (0xfa800225eed0)
    Offset     Content
-------------- -------
           0x0 Data:0x0
           0x8 Data:0xfa8001851a70 Tag:Win\xe4 @0xfa8001851a70
          0x10 Data:0xfa80022c4b30 Tag:Des\xeb @0xfa80022c4b30
          0x18 Data:0xf960003af340 Const:win32k!gTermIO
          0x20 Data:0x0
          0x28 Data:0xf900c015c9f0 Tag:Uskb @0xf900c015c9f0
          0x90 Data:0x0
          0x98 Data:0x0
          0xa0 Data:0x6f6d02580000
          0xa8 Data:0x28b25eb
          0xb0 Data:0xfa800225f010 Empty Tag:moni @0xfa800225f010
          0xb8 Data:0xfa8000927e70 Tag:FxDr @0xfa8000927e70
          0xc0 Data:0xf88000f0c4b0
          0xc8 Data:0x303401002
          0xd0 Data:0xfa8000927e70 Tag:FxDr @0xfa8000927e70
          0xd8 Data:0x10c1a
          0xe0 Data:0xfa80009437b0 _LIST_ENTRY @0xfa80009437b0 Tag:moni @0xfa80009437b0
          0xe8 Data:0xfa8000920460
          0xf0 Data:0x0
          0xf8 Data:0xfa8000919e70 Tag:moni @0xfa8000919e70
```

```
[1] win7.elf.E01 07:21:00> win32k_autodetect
-------------------------> win32k_autodetect()
DEBUG:root:Listed 41 processes using PsActiveProcessHead
DEBUG:root:Listed 37 processes using CSRSS
DEBUG:root:Listed 41 processes using PspCidTable
DEBUG:root:Listed 39 processes using Sessions
DEBUG:root:Listed 40 processes using Handles
DEBUG:root:Switching to process context: System (Pid 4@0xfa80008959e0)
DEBUG:root:Switching to process context: svchost.exe (Pid 236@0xfa80024f85d0)
DEBUG:root:Checking tagWINDOWSTATION at 0xfa800225ef60
DEBUG:root:Unhandled field 0x0, ['Data:0x0']
DEBUG:root:Detected field rpwinstaNext: ['Data:0xfa8001851a70', 'Tag:Win\xe4', '@0xfa8001851a70'] @ 0x8
DEBUG:root:Detected field rpdeskList: ['Data:0xfa80022c4b30', 'Tag:Des\xeb', '@0xfa80022c4b30'] @ 0x10
DEBUG:root:Detected field pTerm: ['Data:0xf960003af340', u'Const:win32k!gTermIO'] @ 0x18
DEBUG:root:Unhandled field 0x20, ['Data:0x0']
DEBUG:root:Unhandled field 0x28, ['Data:0xf900c015c9f0', 'Tag:Uskb', '@0xf900c015c9f0']
DEBUG:root:Unhandled field 0x30, ['Data:0x0']
DEBUG:root:Unhandled field 0x38, ['Data:0x0']
DEBUG:root:Unhandled field 0x40, ['Data:0x0']
DEBUG:root:Unhandled field 0x48, ['Data:0x0']
DEBUG:root:Unhandled field 0x50, ['Data:0x0']
DEBUG:root:Unhandled field 0x58, ['Data:0x0']
DEBUG:root:Unhandled field 0x60, ['Data:0x0']
DEBUG:root:Unhandled field 0x68, ['Data:0x0']
DEBUG:root:Unhandled field 0x70, ['Data:0x0']
DEBUG:root:Detected field pGlobalAtomTable: ['Data:0xf8a002871020', 'Tag:AtmT', '@0xf8a002871020'] @ 0x78
DEBUG:root:Unhandled field 0x80, ['Data:0x0']
DEBUG:root:Unhandled field 0x88, ['Data:0x0']
DEBUG:root:Unhandled field 0x90, ['Data:0x0']
DEBUG:root:Unhandled field 0x98, ['Data:0x0']
DEBUG:root:Unhandled field 0xa0, ['Data:0x6f6d02580000']
DEBUG:root:Unhandled field 0xa8, ['Data:0x28b25eb']
DEBUG:root:Unhandled field 0xb0, ['Data:0xfa800225f010', 'Empty', 'Tag:moni', '@0xfa800225f010']
DEBUG:root:Unhandled field 0xb8, ['Data:0xfa8000927e70', 'Tag:FxDr', '@0xfa8000927e70']
DEBUG:root:Unhandled field 0xc0, ['Data:0xf88000f0c4b0']
DEBUG:root:Unhandled field 0xc8, ['Data:0x303401002']
DEBUG:root:Unhandled field 0xd0, ['Data:0xfa8000927e70', 'Tag:FxDr', '@0xfa8000927e70']
DEBUG:root:Unhandled field 0xd8, ['Data:0x10c1a']
DEBUG:root:Unhandled field 0xe0, ['Data:0xfa80009437b0', '_LIST_ENTRY', '@0xfa80009437b0', 'Tag:moni', '@0xfa80009437b0']
DEBUG:root:Unhandled field 0xe8, ['Data:0xfa8000920460']
DEBUG:root:Unhandled field 0xf0, ['Data:0x0']
DEBUG:root:Unhandled field 0xf8, ['Data:0xfa8000919e70', 'Tag:moni', '@0xfa8000919e70']
DEBUG:root:Unhandled field 0x100, ['Data:0xfa8000919e90', '_LIST_ENTRY', '@0xfa8000919e90', 'Tag:moni', '@0xfa8000919e90']
DEBUG:root:Unhandled field 0x108, ['Data:0xfa8000919e90']
```

Google

```
************ Struct tagWINDOWSTATION ************
       field              offset     Definition
-------------------- ------------- ----------
rpwinstaNext                   0x8 ['Pointer', {'target': 'tagWINDOWSTATION'}]
rpdeskList                    0x10 ['Pointer', {'target': 'tagDESKTOP'}]
pTerm                         0x18 ['Pointer', {'target': 'tagTERMINAL'}]
pGlobalAtomTable              0x78 ['Pointer', {'target': '_RTL_ATOM_TABLE'}]

************** Struct tagTHREADINFO **************
       field              offset     Definition
-------------------- ------------- ----------
pEThread                       0x0 ['Pointer', {'target': '_ETHREAD'}]
GdiTmpTgoList                 0x50 ['_LIST_ENTRY']
ppi                          0x158 ['Pointer', {'target': 'tagPROCESSINFO'}]
pq                           0x160 ['Pointer', {'target': 'tagQ'}]
spklActive                   0x168 ['Pointer', {'target': 'tagKL'}]
rpdesk                       0x178 ['Pointer', {'target': 'tagDESKTOP'}]
PtiLink                      0x260 ['_LIST_ENTRY']

*************** Struct tagDESKTOP ***************
       field              offset     Definition
-------------------- ------------- ----------
rpdeskNext                    0x18 ['Pointer', {'target': 'tagDESKTOP'}]
rpwinstaParent                0x20 ['Pointer', {'target': 'tagWINDOWSTATION'}]
hsectionDesktop               0x78 ['Pointer', {'target': '_SECTION_OBJECT'}]
PtiList                       0xa8 ['_LIST_ENTRY']
```

2: Reverse Engineering code

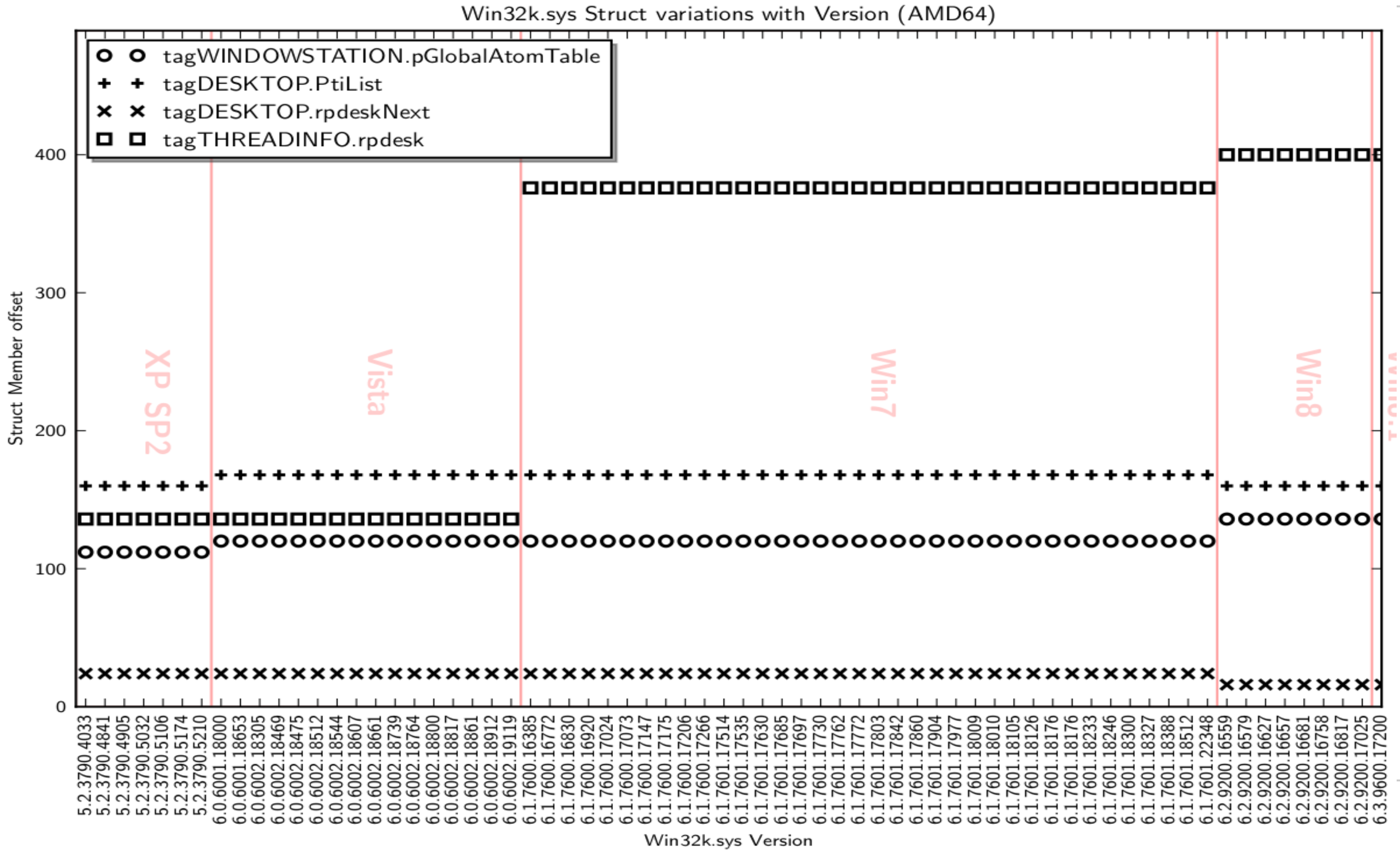
- We need a reproducible and robust reverse engineering method
  - Expert makes the initial reversing analysis
  - Machine parseable method of documenting the finding.
  - Repeatable analysis on similar code variants.

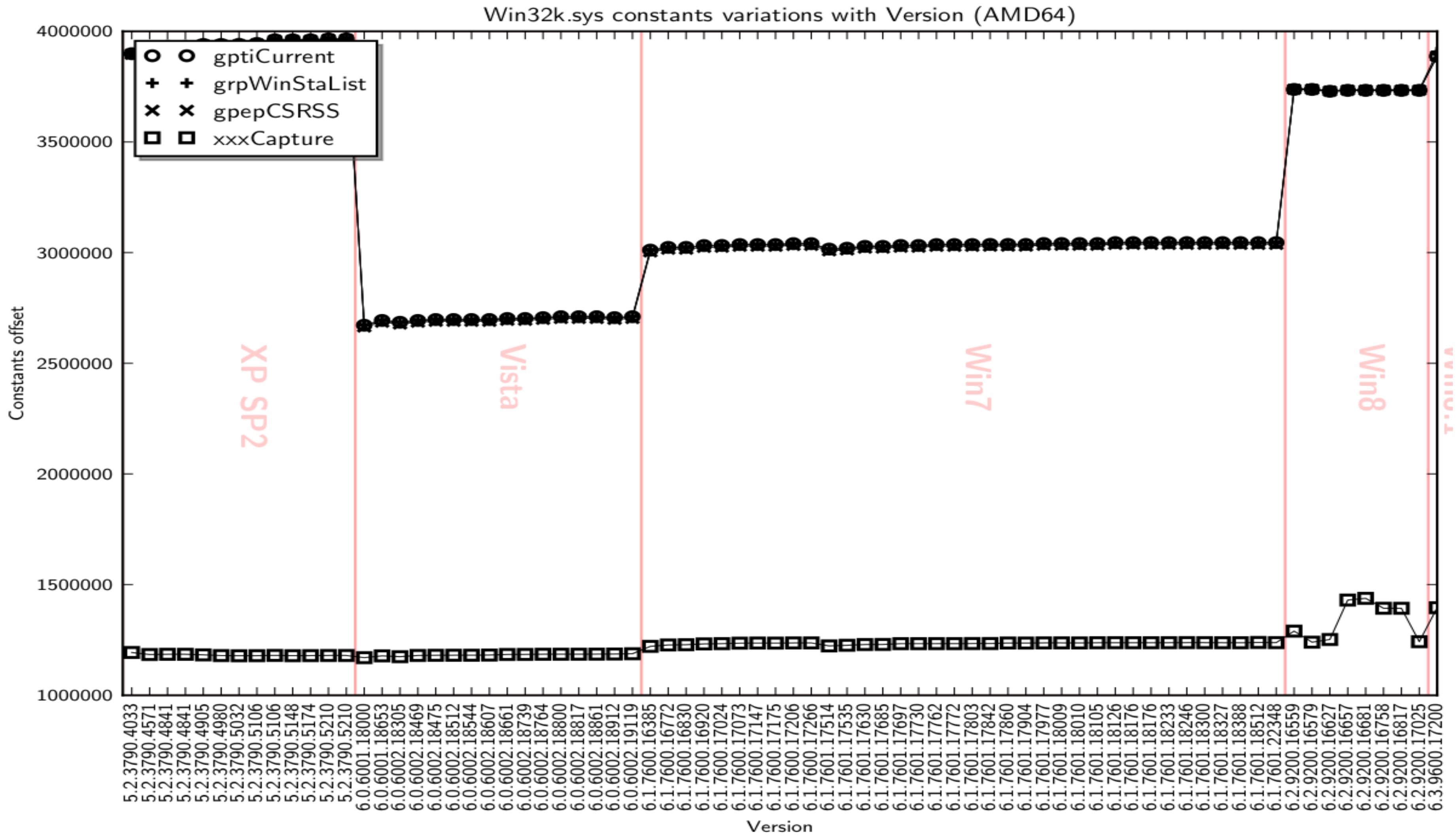
```yaml
tagDESKTOP:
  PtiList:
  - - Disassembler
    - rules:
      - MOV $var1, *grpdeskRitInput
      - TEST $var1, $var1
      - MOV $var1, [$var1+$rpwinstaParent]
      - MOV $pdesk, [$var1+$rpdeskList]
      - LEA *, [$pdesk+$out]
      start: win32k!SetGlobalCursorLevel
      target: Pointer
      max_separation: 300
```

```yaml
pheapDesktop:
- - Disassembler
  - rules:
    - MOV $var1, [*+$out]
    - CALL *RtlAllocateHeap
    start: win32k!DesktopAlloc
    target: Pointer

rpdeskNext:
- - Disassembler
  - rules:
    - MOV $var1, [$var2+$out]
    - TEST $var1, $var1
    - JZ *
    - MOV *CX, $var1
    - CALL *ObQueryNameInfo
    start: win32k!ParseDesktop
    target: Pointer
```

```
[1] win7.elf.E01 09:39:47> print session.profile.tagDESKTOP()
ERROR:root:Failed to find match for tagDESKTOP.rpdeskNext.
DEBUG:root:Unable to find tagDESKTOP.rpdeskNext via Disassemble win32k!ParseDesktop
DEBUG:root:Found match for tagDESKTOP.rpdeskNext
DEBUG:root:.. 0xf960001c4545           0x4d 488b4e18             MOV RCX, [RSI+0x18]
DEBUG:root:.. 0xf960001c4549           0x51 4885c9               TEST RCX, RCX
DEBUG:root:.. 0xf960001c4553           0x5b ff15c7ba1a00         CALL QWORD [RIP+0x1abac7]      0xfffff80002695fe0 win32k!imp_ObfDereferenceObject ->
Object
DEBUG:root:Found match for tagDESKTOP.PtiList
DEBUG:root:  0xf96000206452            0x6 488b05579c1a00       MOV RAX, [RIP+0x1a9c57]        0xfffffa80022c4b30 win32k!grpdeskRitInput
DEBUG:root:  0xf9600020645b            0xf 4885c0               TEST RAX, RAX
DEBUG:root:  0xf96000206460           0x14 488b4020             MOV RAX, [RAX+0x20]
DEBUG:root:  0xf96000206464           0x18 488b5010             MOV RDX, [RAX+0x10]
DEBUG:root:.. 0xf9600020646a           0x1e 4c8d82a8000000       LEA R8, [RDX+0xa8]
ERROR:root:Failed to find match for tagDESKTOP.rpwinstaParent.
DEBUG:root:Unable to find tagDESKTOP.rpwinstaParent via Disassemble win32k!SetGlobalCursorLevel
DEBUG:root:Found match for tagDESKTOP.pheapDesktop
DEBUG:root:. 0xf960001924a5           0x11 488b8980000000       MOV RCX, [RCX+0x80]
DEBUG:root:. 0xf960001924b1           0x1d ff1521e01d00         CALL QWORD [RIP+0x1de021]      0xfffff8000265e840 win32k!imp_RtlAllocateHeap -> nt!Rt
[tagDESKTOP tagDESKTOP] @ 0x00000000
  0x00 rpwinstaParent <None Pointer to [0x00000000] (rpwinstaParent)>
  0x18 rpdeskNext     <None Pointer to [0x00000000] (rpdeskNext)>
  0x80 pheapDesktop   <None Pointer to [0x00000000] (pheapDesktop)>
  0xA8 PtiList        <None Pointer to [0x00000000] (PtiList)>
```

Repeat analysis with undocumented structures





Win32k.sys constants variations with Version (AMD64)
gptiCurrent
grpWinStaList
gpepCSRSS
xxxCapture
Constants offset
4000000
3500000
3000000
2500000
2000000
1500000
1000000
XP SP2
Vista
Win7
Win8
Version
5.2.3790.4033
5.2.3790.4571
5.2.3790.4841
5.2.3790.4841
5.2.3790.4905
5.2.3790.4980
5.2.3790.5032
5.2.3790.5106
5.2.3790.5106
5.2.3790.5148
5.2.3790.5174
5.2.3790.5210
5.2.3790.5210
6.0.6001.18000
6.0.6001.18653
6.0.6002.18305
6.0.6002.18469
6.0.6002.18475
6.0.6002.18512
6.0.6002.18544
6.0.6002.18607
6.0.6002.18661
6.0.6002.18739
6.0.6002.18764
6.0.6002.18800
6.0.6002.18817
6.0.6002.18861
6.0.6002.18912
6.0.6002.19119
6.1.7600.16385
6.1.7600.16772
6.1.7600.16830
6.1.7600.16920
6.1.7600.17024
6.1.7600.17073
6.1.7600.17147
6.1.7600.17175
6.1.7600.17206
6.1.7600.17266
6.1.7601.17514
6.1.7601.17535
6.1.7601.17630
6.1.7601.17685
6.1.7601.17697
6.1.7601.17730
6.1.7601.17762
6.1.7601.17772
6.1.7601.17803
6.1.7601.17842
6.1.7601.17860
6.1.7601.17904
6.1.7601.17977
6.1.7601.18009
6.1.7601.18010
6.1.7601.18105
6.1.7601.18126
6.1.7601.18176
6.1.7601.18176
6.1.7601.18233
6.1.7601.18246
6.1.7601.18300
6.1.7601.18327
6.1.7601.18388
6.1.7601.18512
6.1.7601.22348
6.2.9200.16559
6.2.9200.16579
6.2.9200.16627
6.2.9200.16657
6.2.9200.16681
6.2.9200.16758
6.2.9200.16817
6.2.9200.17025
6.3.9600.17200

- Scanning for constants in undocumented code is extremely difficult.
  - Often requires the automated disassembly of function call sites.
  - Often loses context (e.g. which session does this win32k object belong to?).
  - Quite slow.
- We want to be able to use known versions
  - Just extract constants from the PDB files.

What does a Rekall profile look like?

```
{ "$CONSTANTS": {
 "CmNtCSDVersion": 718856,
...
 "$ENUMS": {
  "BUS_QUERY_ID_TYPE": {
   "0": "BusQueryDeviceID",
   "1": "BusQueryHardwareIDs",
...
 "$FUNCTIONS": {
  "ADD_MAP_REGISTERS": 606670,
...
 "$METADATA": {
  "ProfileClass": "Nt",
  "arch": "I386"
...
 "$STRUCTS": {
  "BATTERY_REPORTING_SCALE": [8, {
   "Capacity": [4, ["unsigned long", {}]], ...
```

- File is a JSON data structure.
- Divided into Sections:
  - $CONSTANTS
  - $FUNCTIONS
  - $METADATA
  - $STRUCTS

$STRUCT section.

```
{ "_EPROCESS": [624, {
  "AccountingFolded": [548, ["BitField", {
   "end_bit": 2,
   "start_bit": 1,
   "target": "unsigned long"
   }]],
  "ActiveProcessLinks": [160, ["_LIST_ENTRY", {}]],
  "ActiveThreads": [376, ["unsigned long", {}]],
  "AddressCreationLock": [232, ["_EX_PUSH_LOCK", {}]],
  "AddressSpaceInitialized": [552, ["BitField", {
   "end_bit": 12,
   "start_bit": 10,
   "target": "unsigned long"
   }]],
  "AffinityPermanent": [548, ["BitField", {
   "end_bit": 19,
   "start_bit": 18,
   "target": "unsigned long"...
```

Struct Size

Member Offset

Member Type

Struct Name

Arguments to the member type.

$CONSTANTS and $FUNCTIONS

```
"NtAlpcSendWaitReceivePort": 2207436,
 "NtAlpcSetInformation": 1805611,
 "NtApphelpCacheControl": 2308968,
 "NtAreMappedFilesTheSame": 2367400,
 "NtAssignProcessToJobObject": 1912487,
 "NtBuildGUID": 411132,
 "NtBuildLab": 410688,
 "NtBuildLabEx": 410912, ...
```

Constant name

Constant offset (Relative to the kernel base).

- These addresses come directly from Microsoft Debugging symbols.
  - Identical to the way the kernel debugger works.
  - No need to scan, guess or otherwise deduce symbol addresses.

Rekall Profiles - JSON files

- A profile file is a data structure which represents all the information needed to parse OS specific memory.
  - Files are stored in the public profile repository:
    - http://profiles.rekall-forensic.com
  - Windows Profiles are identified by GUID.

Revision c39b14f8dca9: /nt/GUID

[Project Page]

- ..
- 00625D7D36754CBEBA4533BA9A0F3FE22.gz
- 0100FCDAFD4049B8B06005EC07705A1F2.gz
- 01DDCBD82AEB46BEAFCDC6A409E3B1D31.gz
- 01DF28C698D84DEBB1A74254C3AF800E2.gz
- 03185083233249D9BB747EA777B80C982.gz
- 04FB9A156FF44ECCA6EBCAE9617D8DB73.gz
- 05A6F49C5DD848FF983459421A78E1232.gz

Profiles for nt kernel are stored here.

Every single kernel build has a unique GUID.

Profile Indexes

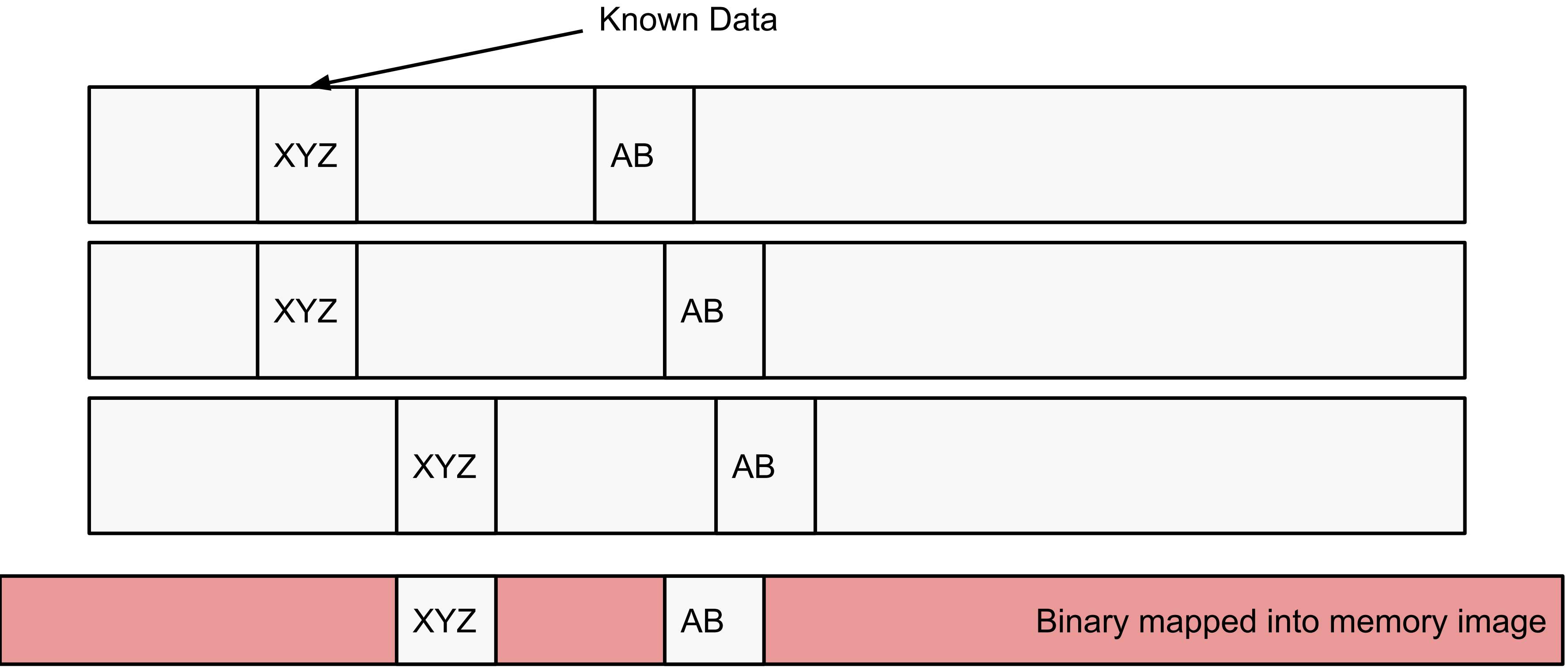
Problem statement: Set membership:

- Given a set of binaries, is this binary in the set, and which one is it?

Solution:

- Generate a sample of data points in each binary and build a decision tree.

Conceptual overview



Potential Complications

- Not all pages in binary are always readable:
  - Must discount comparison points in unreadable pages -> weakens signatures.
- How many points should we use?
- Sometimes we do not have the actual binary - we only have the binary GUID.
  - Deduce data in binary purely from symbol information:
    - Functions have known preamble.
    - String constants have debug symbols.

```
path: nt
symbols:
  -
    name: "str:KernelSpace"
    data:
      - "str:KernelSpace"

  -
    name: "str:ZwQueryInformationFile"
    data:
      - "str:ZwQueryInformationFile"

   name: ExEnumHandleTable
    data: ["90"]
    shift: -1

  -
    name: FsRtlAllocateFileLock
    data: ["90"]
    shift: -1
```

Symbol Name (Usually mangled)

Probably what the address contains.

Before each function - NOP slide.

Result index

Profile Name

Offset in binary.

Expected data.

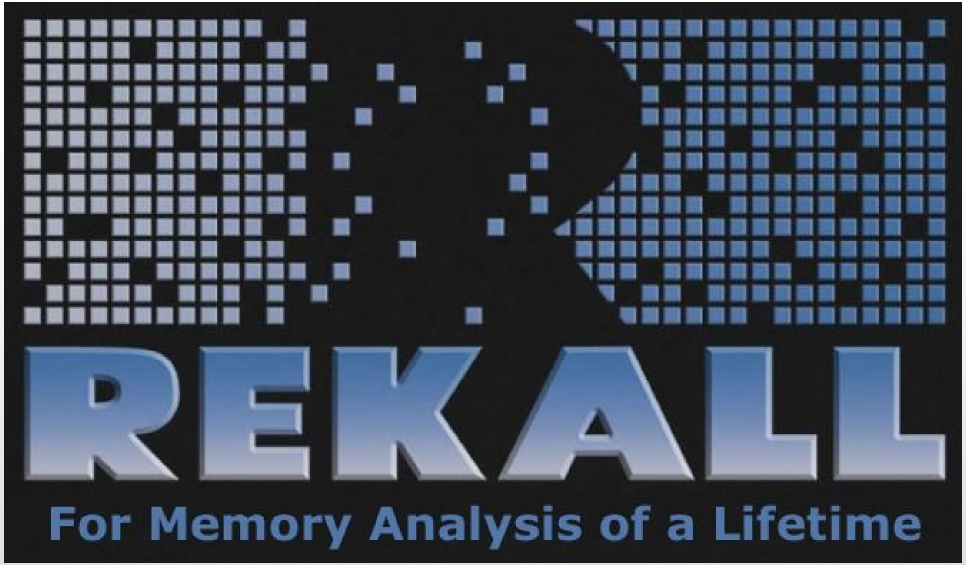
{
 "$INDEX": {
  "nt/GUID/00625D7D36754CBEBA4533BA9A0F3FE22": [[2038160, ["4b65726e656c5370616365"]], [3601204, ["4952505f4d4e5f51554552595f4445564943455f54455854"]], [253980, ["410\
0500050005f004e0041004d004500"]], [120086, ["90"]], [2559256, ["90"]], [137962, ["90"]], [2500200, ["90"]], [2569084, ["90"]], [206055, ["90"]], [630264, ["90"]], [10\
4589, ["90"]]],
  "nt/GUID/0100FCDAFD4049B8B06005EC07705A1F2": [[463544, ["5a7751756572794e666f726d6174696f6e46696c65"]], [2197832, ["4952505f4d4e5f51554552595f4445564943455f544558\
54"]], [206384, ["4100500050005f004e0041004d004500"]], [299937, ["90"]], [1174215, ["90"]], [113888, ["90"]], [1339912, ["90"]], [1746854, ["90"]], [236722, ["90"]], \
[264355, ["90"]]],

Conclusions

- Validating assumptions about kernel versions.
  - Will our analysis work in every case?
    - Maybe not
- Develop methods for automated reverse engineering
  - Helps to document expert effort.
  - Helps to repeat on many samples.
- Towards fully automated Linux profile generation!
  - Given a binary kernel image, calculate the correct profile automatically.

http://www.rekall-forensic.com/

Sorry, Quaid. Your whole life is just a dream.



See you at the party, Richter!