DFRWS USA 2016

Robust Bootstrapping Memory Analysis against Anti-forensics

Kyoungho Lee, Hyunuk Hwang, Kibom Kim, and Bongnam Noh

2016-08-08

Kyoungho Lee

koungholee@gmail.com

CHONNAM NATIONAL UNIVERSITY | SYSTEM SECURITY RESEARCH CENTER

Content

1. **Background**
2. **Memory analysis based on KiInitialPCR**
   1. Carving KiInitialPCR
   2. Analysis based on KiInitialPCR
3. **Conclusion**

Background

Memory Forensics

- **Forensic analysis of a computer's memory dump**
  - Acquiring physical memory from live system
  - Collecting evidence from memory image

- **Bootstrapping analysis**
  - OS fingerprinting
    - Accurate structure layout, analysis algorithm
  - Acquiring directory table base
    - Translating virtual address to physical address
  - Obtaining kernel objects
    - Kernel data such as process related information

  - Reconstruct live system state from memory image

Anti memory forensics

▪ Anti analysis

- Focuses on making investigators fail to collect volatile evidence by modifying values used in the memory analysis
- **One-byte abort factor**
  - modify fragile signatures to block the analysis algorithm
- Semantic Value Manipulation (SVM)
  - compromise kernel data structure’s field which has a semantic value
- Attention-Deficit-Disorder (ADD)
  - construct fake kernel objects to increase the analysis time

Before abort factor attack

```
Volatility Foundation Volatility Framework 2.5
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64
---------- -------------------- ------ ------ ------ -------- ------ ------
0x853f6908 System                    4      0     96      520 ------      0
0x859fb1c0 smss.exe                268      4      2       30 ------      0
0x854cf480 csrss.exe               344    332      9      518      0      0
0x86b96148 csrss.exe               436    428     10      258      1      0
0x86b98720 wininit.exe             444    332      4       78      0      0
0x86eb1d28 winlogon.exe            488    428      6      118      1      0
```

After abort factor attack

```
Volatility Foundation Volatility Framework 2.5
No suitable address space mapping found
Tried to open image as:
 MachOAddressSpace: mac: need base
 LimeAddressSpace: lime: need base
 WindowsHiberFileSpace32: No base Address Space
 WindowsCrashDumpSpace64BitMap: No base Address Space
 VMWareMetaAddressSpace: No base Address Space
```

Against anti-forensics

- **Profile indexing method**
  - Arbitrarily choose addresses from debugging symbols
  - Generate profiles composed of the offsets
  - Obtain DTB and find kernel base
  - Determine kernel version by comparing values at addresses
- **Limitation**
  - Values can be modified despite randomly choosing
  - Weakness to get kernel base (PE signature)

Profile RVA Kernel Base NOP instruction

```
nt/GUID/74877E6D37F846E693D3B86851AC73332 matched offset 0x45d582+0xf80002c18000=0xf80003075582 ('₩x90')
nt/GUID/74877E6D37F846E693D3B86851AC73332 matched offset 0x3ab071+0xf80002c18000=0xf80002fc3071 ('₩x90')
nt/GUID/74877E6D37F846E693D3B86851AC73332 matches 2/12 comparison points
nt/GUID/5541D5331BD348C699EC41CFDE194B112 matched offset 0x1a2c3+0xf80002c18000=0xf80002c322c3 ('₩x90')
nt/GUID/5541D5331BD348C699EC41CFDE194B112 matches 1/12 comparison points
nt/GUID/F7BEC858A4C3441B8C80F1E9994EC09E2 matched offset 0x1ddeb+0xf80002c18000=0xf80002c35deb ('₩x90')
nt/GUID/F7BEC858A4C3441B8C80F1E9994EC09E2 matches 1/13 comparison points
nt/GUID/918329E2ABE74926B63736573F7CB2A31 matched offset 0xaf9af+0xf80002c18000=0xf80002cc79af ('₩x90')
nt/GUID/918329E2ABE74926B63736573F7CB2A31 matches 1/10 comparison points
```

Assessments of anti-forensics

▪ Attack Targets

Target	Uses
System EPROCESS	used to identify the OS version and to obtain the DTB
Idle EPROCESS	used to obtain the DTB
KDBG structure	used to identify the OS version
RSDS region	**used to identify the kernel build version, including the OS version**
Kernel PE signature	**used to find kernel base for OS fingerprinting**
Comparison points	**used to identify the kernel build version, including the OS version**

▪ How we attack the targets?

- Modify DispatcherHeader, ImageFileName, OwnerTag (by abort factor)
- Modify RSDS region, PE signature of the kernel executable and part of comparison points
- All values at these location don't generate system crashes

Assessments of anti-forensics

▪ Evaluation environment

- Windows 7 SP1 64-bit on Vmware (fully updated)
- Extracting process list (not carving)
  - common function for OS fingerprinting and acquring DTB
  - important function to enable process deep analysis

▪ Results

- All tested tools can be defeated with three bytes overwritten

Memory Modification Target	volatility 2.5	memoryze 3.0	rekall 1.4.1 (RSDS)	rekall 1.4.1 (nt index)
Idle Process	×	o	×	o
System Process	o	×	o	o
KDBG	×	o	o	o
RSDS	o	o	×	o
PE signatures	o	o	o	×
Comparison points	o	o	o	×

The symbol O indicates that the tool successfully extracts the process list
The symbol × indicates that the tool fails to analyze the image

Challenges

- **Robust fields are needed for**
  - OS fingerprinting, Acquiring DTB, Collecting kernel objects

- **Following structure is needed**
  - same structure layout, carving rule
  - robust fields for OS fingerprinting
  - robust fields containing DTB
  - robust fields to access kernel global variables

- **We find it!**
  - KiInitialPCR, which is first instance of KPCR structre

Memory analysis based on KiInitialPCR

Feature of KiInitialPCR

- **KPCR structure**
  - The number of KPCR structures is equal to the number of processors
  - Same structure layout per machine bit
  - Self-reference field named as SelfPcr(or Self on 64bit)
  - Cr3 field has the DirectoryTableBase
    - used to find KPCR instance (Ruichao Zhang et al. suggest)

```
nt!_KPCR
   …
   +0x01c SelfPcr          : Ptr32 _KPCR        Self-reference
   +0x020 Prcb             : Ptr32 _KPRCB
   +0x038 IDT              : Ptr32 _KIDTENTRY
   +0x03c GDT              : Ptr32 _KGDTENTRY
   +0x040 TSS              : Ptr32 _KTSS
   …
   +0x120 PrcbData         : _KPRCB
      …
      +0x00c IdleThread : Ptr32 _KTHREAD
      +0x018 ProcessorState   : _KPROCESSOR_STATE
         +0x000 ContextFrame     : _CONTEXT
         +0x2cc SpecialRegisters : _KSPECIAL_REGISTERS
            +0x000 Cr0              : Uint4B
            +0x004 Cr2              : Uint4B
            +0x008 Cr3              : Uint4B     DirectoryTableBase
      +0x3cc Number : Uint4B
```

Feature of KiInitialPCR

- **KPCR structure**
  - Same structure layout ?

```
0: kd> dt _KPCR
nt!_KPCR
   +0x000 NtTib            : _NT_TIB
   +0x000 GdtBase          : Ptr64 _KGDTENTRY64
   +0x008 TssBase          : Ptr64 _KTSS64
   +0x010 UserRsp          : Uint8B
   +0x018 Self             : Ptr64 _KPCR
   +0x020 CurrentPrcb      : Ptr64 _KPRCB
   +0x028 LockArray        : Ptr64 _KSPIN_LOCK_QUEUE
   +0x030 Used_Self        : Ptr64 Void
   +0x038 IdtBase          : Ptr64 _KIDTENTRY64
   +0x040 Unused           : [2] Uint8B
   +0x050 Irql             : UChar
   +0x051 SecondLevelCacheAssociativity : UChar
   +0x052 ObsoleteNumber   : UChar
   +0x053 Fill0            : UChar
   +0x054 Unused0          : [3] Uint4B
   +0x060 MajorVersion     : Uint2B
   +0x062 MinorVersion     : Uint2B
   +0x064 StallScaleFactor : Uint4B
   +0x068 Unused1          : [3] Ptr64 Void
   +0x080 KernelReserved   : [15] Uint4B
   +0x0bc SecondLevelCacheSize : Uint4B
   +0x0c0 HalReserved      : [16] Uint4B
   +0x100 Unused2          : Uint4B
   +0x108 KdVersionBlock   : Ptr64 Void
   +0x110 Unused3          : Ptr64 Void
   +0x118 PcrAlign1        : [24] Uint4B
   +0x180 Prcb             : _KPRCB
```

identical

```
0: kd> dt _KPCR
nt!_KPCR
   +0x000 NtTib            : _NT_TIB
   +0x000 GdtBase          : Ptr64 _KGDTENTRY64
   +0x008 TssBase          : Ptr64 _KTSS64
   +0x010 UserRsp          : Uint8B
   +0x018 Self             : Ptr64 _KPCR
   +0x020 CurrentPrcb      : Ptr64 _KPRCB
   +0x028 LockArray        : Ptr64 _KSPIN_LOCK_QUEUE
   +0x030 Used_Self        : Ptr64 Void
   +0x038 IdtBase          : Ptr64 _KIDTENTRY64
   +0x040 Unused           : [2] Uint8B
   +0x050 Irql             : UChar
   +0x051 SecondLevelCacheAssociativity : UChar
   +0x052 ObsoleteNumber   : UChar
   +0x053 Fill0            : UChar
   +0x054 Unused0          : [3] Uint4B
   +0x060 MajorVersion     : Uint2B
   +0x062 MinorVersion     : Uint2B
   +0x064 StallScaleFactor : Uint4B
   +0x068 Unused1          : [3] Ptr64 Void
   +0x080 KernelReserved   : [15] Uint4B
   +0x0bc SecondLevelCacheSize : Uint4B
   +0x0c0 HalReserved      : [16] Uint4B
   +0x100 Unused2          : Uint4B
   +0x108 KdVersionBlock   : Ptr64 Void
   +0x110 Unused3          : Ptr64 Void
   +0x118 PcrAlign1        : [24] Uint4B
   +0x180 Prcb             : _KPRCB
```

it has same layout?

Windows 7 32-bit

Windows 10 32-bit

Feature of KiInitialPCR

▪ KPCR structure

- Same structure layout ?

```
0: kd> dt _KPRCB
nt!_KPRCB
   +0x000 MxCsr            : Uint4B
   +0x004 LegacyNumber     : UChar
   +0x005 ReservedMustBeZero : UChar
   +0x006 InterruptRequest : UChar
   +0x007 IdleHalt         : UChar
   +0x008 CurrentThread    : Ptr64 _KTHREAD
   +0x010 NextThread       : Ptr64 _KTHREAD
   +0x018 IdleThread       : Ptr64 _KTHREAD
   +0x020 NestingLevel     : UChar
   +0x021 PrcbPad00        : [3] UChar
   +0x024 Number           :
   +0x028 RspBase          :
   +0x030 PrcbLock         :
   +0x038 PrcbPad01        :
   +0x040 ProcessorState   :
   +0x5f0 CpuType          :
   +0x5f1 CpuID            :
   +0x5f2 CpuStep          :
   +0x5f2 CpuStepping      :
   +0x5f3 CpuModel         : UChar
   +0x5f4 MHz              : Uint4B
   +0x5f8 HalReserved      : [8] Uint8B
   +0x638 MinorVersion     : Uint2B
   +0x63a MajorVersion     : Uint2B
   +0x63c BuildType        : UChar
   +0x63d CpuVendor        : UChar
   +0x63e CoresPerPhysicalProcessor : UChar
   +0x63f LogicalProcessorsPerCore : UChar
   +0x640 ApicMask         :
   +0x644 CFlushSize       :
   +0x648 AcpiReserved     :
   +0x650 InitialApicId    : Uint4B
   +0x654 Stride           : Uint4B
   +0x658 Group            : Uint2B
   +0x660 GroupSetMember   : Uint8B
```

Windows 7 32-bit

new fields are added

same offset and field name

offset and field are different

```
0: kd> dt _KPRCB
nt!_KPRCB
   +0x000 MxCsr            : Uint4B
   +0x004 LegacyNumber     : UChar
   +0x005 ReservedMustBeZero : UChar
   +0x006 InterruptRequest : UChar
   +0x007 IdleHalt         : UChar
   +0x008 CurrentThread    : Ptr64 _KTHREAD
   +0x010 NextThread       : Ptr64 _KTHREAD
   +0x018 IdleThread       : Ptr64 _KTHREAD
   +0x020 NestingLevel     : UChar
   +0x021 ClockOwner       : UChar
   +0x022 PendingTickFlags : UChar
   +0x022 PendingTick      : Pos 0, 1 Bit
   +0x022 PendingBackupTick : Pos 1, 1 Bit
   +0x023 IdleState        : UChar
   +0x024 Number           : Uint4B
   +0x028 RspBase          : Uint8B
   +0x030 PrcbLock         : Uint8B
   +0x038 PriorityState    : Ptr64 Char
   +0x040 ProcessorState   : _KPROCESSOR_STATE
   +0x5f0 CpuType          : Char
   +0x5f1 CpuID            : Char
   +0x5f2 CpuStep          : Uint2B
   +0x5f2 CpuStepping      : UChar
   +0x5f3 CpuModel         : UChar
   +0x5f4 MHz              : Uint4B
   +0x5f8 HalReserved      : [8] Uint8B
   +0x638 MinorVersion     : Uint2B
   +0x63a MajorVersion     : Uint2B
   +0x63c BuildType        : UChar
   +0x63d CpuVendor        : UChar
   +0x63e CoresPerPhysicalProcessor : UChar
   +0x63f LogicalProcessorsPerCore : UChar
   +0x640 ParentNode       : Ptr64 _KNODE
   +0x648 GroupSetMember   : Uint8B
   +0x650 Group            : UChar
   +0x651 GroupIndex       : UChar
```

Windows 10 32-bit

Feature of KiInitialPCR

- **KiInitialPCR**
  - first instance of KPCR structure
    - KiInitialPCR is a kernel global variable
    - other KPCR structures are in dynamic memory
  - Self-reference field enables us get other kernel global variables
    - By adding offsets from KiInitialPCR, instead of kernel base
    - On only same kernel build version
    - PsActiveProcessHead, PsLoadedModuleList, KDBG and etc.

Carving of KiInitialPCR

- **Robust signature generation for KPCR structure**
  - Fuzzing Stage
    - Find KPCR structure with existing carving rule
    - Rebooting for allocating new KPCR structure
      - KPCR is only allocated by kernel
    - Tested on quad-core CPU for multiple KPCR instance
      - Windows 7, 8, 10 32/64-bit
  - Generated signatures
    - When below fields are mutated, system crashes immediately

Field(32bit/64bit)	32bit	64bit
Prcb/CurrentPrcb	val == SelfPcr + 0×120 && val % 0×20 == 0	val == CurrentPrcb + 0×180 && val % 0×20 == 0
SelfPcr/Self	val == Prcb − 0×120 && val % 0×100 == 0	val == Self − 0×180 && val % 0×100 == 0
GDT/GdtBase	val % 0×1000 == 0	val % 0×1000 == 0
-/LockArray	-	val == CurrentPrcb + 0×670
Union	val != 0 && val >= 0×80000000	val != 0 && val >= 0×FFFF000000000000

Carving of KiInitialPCR

- **Selection of KiInitialPCR**
  - We should choose the KiInitialPCR among the KPCRs being carved
  - Number field
    - Unique ID for processors
      - KiInitialPCR has zero value
    - offset is fixed as 0x3cc in 32-bit and 0x24 in 64-bit
    - Is the Number field is robust?
      - Yes, the system crashes when this field is modified

- **Cr3 field**
  - used for virtual address translation
  - Is the Cr3 field is robust?
    - It doesn’t generate system crashes
    - It is renewed continueously when it is modified

Memory analysis using KiInitialPCR

▪ Identifying OS version

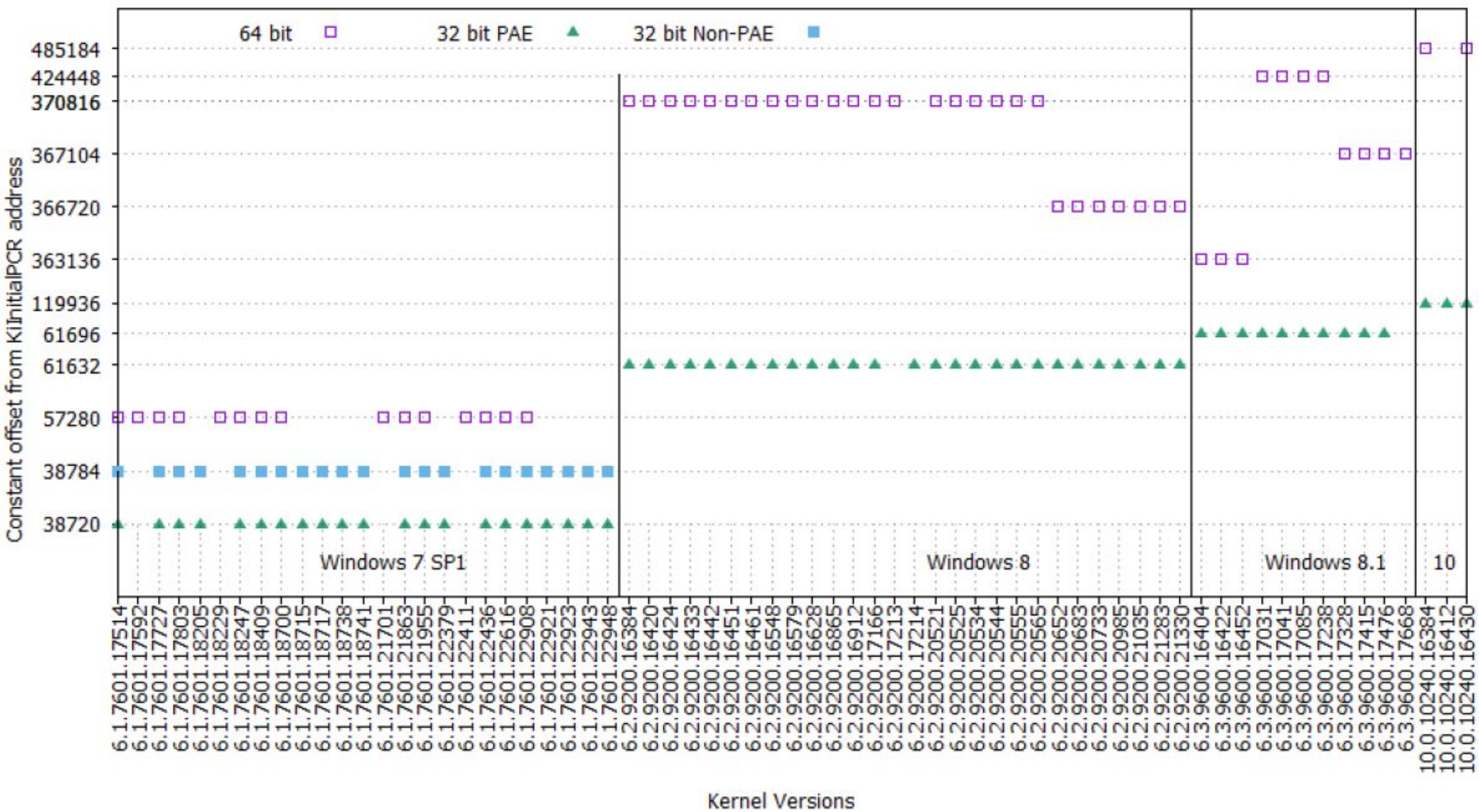
- IdleThread field points to idle thread for each processor
  - KiInitialPCR's *IdleThread* points global variable **KiInitialThread**
  - other KPCR's *IdleThread* points ETHREAD structure in kernel heap memory

- Global variables are located at fixed location from kernel base
  - Relative offsets greatly vary with the kernel build version
  - **Distance between KiInitialThread and KiInitialPCR greatly varies or not?**

- Gathering Windows kernel executables
  - WinSxs Folder on Windows 7/8/10 32/64-bit

Memory analysis using KiInitialPCR

- **Identifying OS version**
  - Offsets between KiInitialPCR and KiInitialThread



Memory analysis using KiInitialPCR

- **Identifying OS version**
  - Are SelfPcr and IdleThread field robust?
    - Zero: null bytes
    - Random: four/eight random bytes
    - Random primitive type: fuzzed using valid pointers to other ethread
    - **Fuzzing with zero, random, random primitive type generates crashes**
  - Is it really robust?
    - Attacker can relocate KiInitialPCR and KiInitialThread
    - Then make pointer fields point new relocated memory!
      - If this attack is possible, our os version signature is weak

Memory analysis using KiInitialPCR

- **Identifying OS version**
  - Relocating KiInitialPCR
    - Copied KiInitialPCR and made the Self field point to the copied one
      - as well as GS:[0x18], GS:[0x20]

```
0: kd> dqs gs:18 L2
002b:00000000`00000018  fffff802`f9122000 nt!KiInitialPCR
002b:00000000`00000020  fffff802`f9122180 nt!KiInitialPCR+0x180
```

    - System has stopped immediately when the pointers were modified
      - Tested on Windows 7, 8, 10 32/64bit
      - Because the processor state is different before and after the copy

Memory analysis using KiInitialPCR

▪ Identifying OS version

- Relocating KiInitialThread
  - Copied KiInitialThread and made the IdleThread field point to the copied
  - System crashes with BSOD after a few minutes
    - On Windows 8, 10 32/64-bit
  - System doesn't crash on Windows 7 32/64-bit

  - We can still identify OS version
    - The remainder of the KiInitialPCR offset is 0×d00 or 0×c00 on Windows 7
    - The remainder is 0×1000 on Windows 8, 8.1 and 10

```
File : D:\ntoskrnlset\Win7SP1x64\amd64_micro
# Found InitialPcr RVA : 0x1f1d00
File : D:\ntoskrnlset\Win7SP1x64\amd64_micro
# Found InitialPcr RVA : 0x1f1d00
File : D:\ntoskrnlset\Win7SP1x64\amd64_micro
# Found InitialPcr RVA : 0x1f1d00
File : D:\ntoskrnlset\Win7SP1x64\amd64_micro
# Found InitialPcr RVA : 0x1f1d00
```

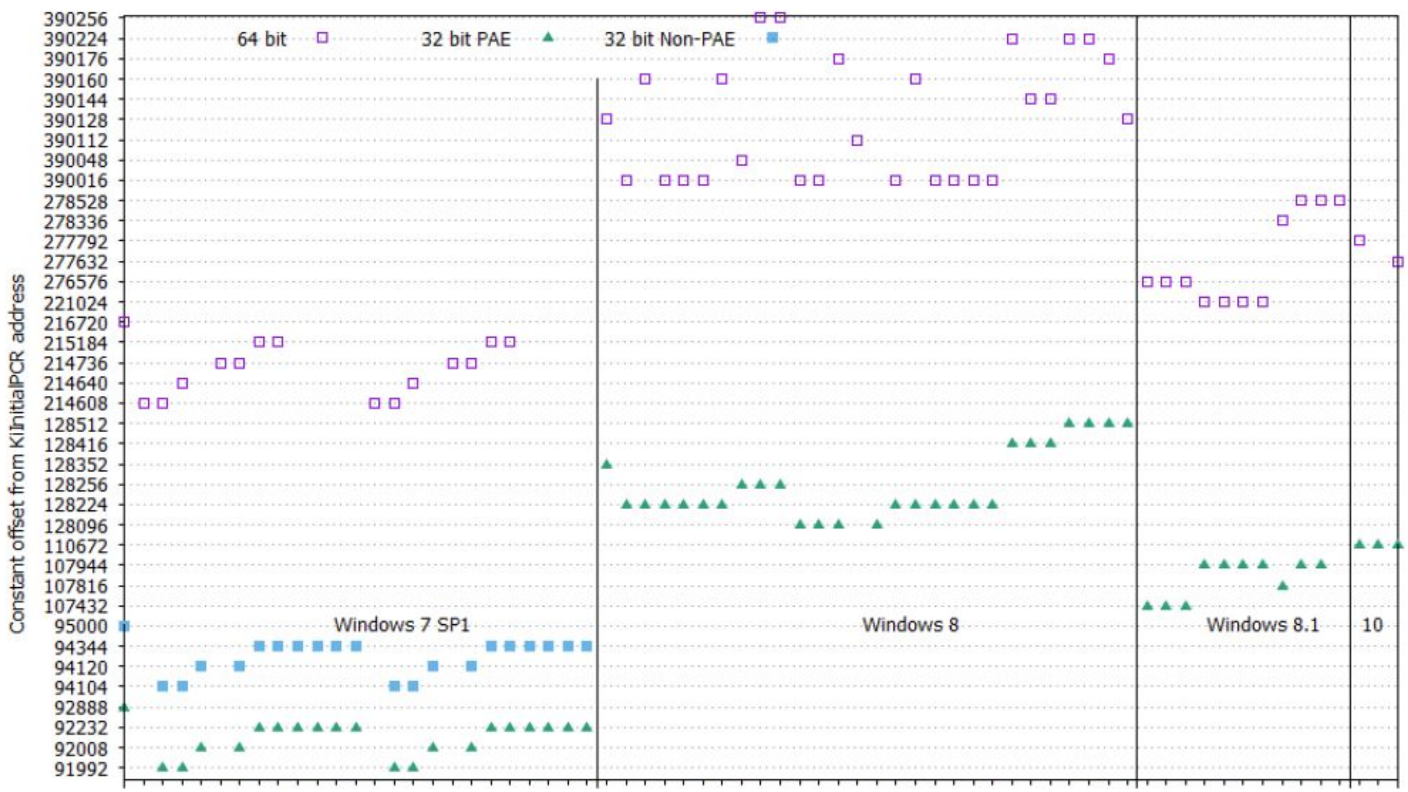
Windows 7

```
File : D:\ntoskrnlset\Win8SP1x64\amd64_mic
# Found InitialPcr RVA : 0x303000
File : D:\ntoskrnlset\Win8SP1x64\amd64_mic
# Found InitialPcr RVA : 0x2ff000
File : D:\ntoskrnlset\Win8SP1x86\x86_micro
# Found InitialPcr RVA : 0x20b000
File : D:\ntoskrnlset\Win8SP1x86\x86_micro
# Found InitialPcr RVA : 0x20b000
```

Windows 8

Memory analysis using KiInitialPCR

- **Process list extraction**
  - Main function of memory forensic tools
    - EPROCESS structure contains thread, module information and etc.
  - Offsets between KiInitialPCR and PsActiveProcessHead



Memory analysis using KiInitialPCR

- **Process list extraction**
  - No direct method to determine a valid PsActiveProcessHead
  - we need to identify whether these offsets are valid process head
    - Finite sets composed of the offsets based on each version signature
    - Cardinalities of these sets are fewer than eight
  - We check whether the list entry is completely traversed or not
  - Also validate all EPROCESS structures by checking robust signatures
    - Use EPROCESS signature in robust signature research

Field	Constraint
Pcb.ReadyListHead.Flink	`val & 0x80000000 == 0x80000000 && val % 0x8 == 0`
Pcb.ThreadListHead.Flink	`val & 0x80000000 == 0x80000000 && val % 0x8 == 0`
WorkingSetLock.Count	`val == 1 && val & 0x1 == 0x1`
Vm.VmWorkingSetList	`val & 0xc0003000 == 0xc0003000 && val % 0x1000 == 0`
VadRoot	`val == 0 || (val & 0x80000000 == 0x80000000 && val % 0x8 == 0)`
Token.Value	`val & 0xe0000000 == 0xe0000000`
AddressCreationLock.Count	`val == 1 && val & 0x1 == 0x1`
VadHint	`val == 0 || (val & 0x80000000 == 0x80000000 && val % 0x8 == 0)`
Token.Object	`val & 0xe0000000 == 0xe0000000`
QuotaBlock	`val & 0x80000000 == 0x80000000 && val % 0x8 == 0`
ObjectTable	`val == 0 || (val & 0xe0000000 == 0xe0000000 && val % 0x8 == 0)`
GrantedAccess	`val & 0x1f07fb == 0x1f07fb`
ActiveProcessLinks.Flink	`val & 0x80000000 == 0x80000000 && val % 0x8 == 0`
Peb	`val == 0 || (val & 0x7ffd0000 == 0x7ffd0000 && val % 0x1000 == 0)`
Pcb.DirectoryTableBase.0	`val % 0x20 == 0`

Implementation

- **Performance**
  - Windows 7 32/64-bit
    - Average analysis time is 2 seconds
    - KiInitialPCR is located at a low physical address
  - Windows 8, 10 32/64-bit 8GB memory
    - Average analysis time is 4 minutes
    - KiInitialPCR is located at the end of the file
  - Volatility's KPCR plugin
    - Finding KPCR structures in reconstructed virtual address space
    - Check the address equality between SelfPcr and the physical offset in every byte
    - Take longer than 1 hours

Conclusion

Conclusion

- **Conclusion**
  - We guarantee the bootstrapping analysis, and they are not subverted by anti-forensic techniques.
  - Our OS fingerprinting and DTB acquisition parts enable precise carving of kernel data structure with accurate structure layout
  - Our robust kernel object listing can find hidden objects by comparing them with carved objects.

- **Future work**
  - Identify an exact kernel version with only robust fields

Any questions?

Q&A

Thank you for coming