Digital Investigation 18 (2016) S23–S32

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

DFRWS USA 2016 — Proceedings of the 16th Annual USA Digital Forensics Research Conference

Robust bootstrapping memory analysis against anti-forensics

Kyoungho Lee [a], Hyunuk Hwang [b, *], Kibom Kim [b], BongNam Noh [a]

[a] System Security Research Center, Chonnam National University, Gwangju, South Korea
[b] The Affiliated Institute of ETRI, Daejeon, South Korea

Keywords:
Windows
Memory analysis
Memory forensics
Robust analysis
OS fingerprinting

A B S T R A C T

Memory analysis is increasingly used to collect digital evidence in incident response. With the fast growth in memory analysis, however, anti-forensic techniques appear to prevent it from performing the bootstrapping steps — operating system (OS) fingerprinting, Directory Table Base (DTB) identification, and obtaining kernel objects. Although most published research works try to solve anti forensics, they deal only with one element among the three steps. Thus, collapse in any of the three steps using the suggested robust algorithms leads to failure in the memory analysis. In this paper, we evaluate the latest memory forensic tools against anti-forensics. Then, we suggest a novel robust algorithm that guarantees the bootstrapping analysis steps. It uses only one kernel data structure called KiInitialPCR, which is a kernel global variable based on the kernel processor control region (KPCR) structure and has many fields with tolerance to mutation. We characterize the robust fields of the KPCR structure to use them for OS fingerprinting, DTB identification, and obtaining kernel objects. Then, we implement the KiInitialPCR-based analysis system. Therefore, we can analyze the compromised memory in spite of the interference of anti-forensics.

Introduction

Memory forensics means the process of acquiring physical memory, which contains volatile data, and collecting evidence from the acquired memory image. Since the Digital Forensic Research Conference memory challenge (Dfrws, 2005) opened in 2005, the extraction of volatile data such as disk encryption key (Mrdovic and Huseinovic, 2011) and user input information (Olajide et al., 2012), as well as the process list (Betz, 2005), has been researched. Memory forensics has become important in digital forensics in that it can extract these volatile data, which is impossible from a hard disk.

Anti-forensic techniques have appeared to prevent memory analysis, due to its importance. According to the definition of anti-forensics (Harris, 2006), anti-forensics to memory analysis is divided into two parts; preventing memory acquisition and memory analysis. In these aspects, research on anti-memory acquisition has progressed (Stüttgen and Cohen, 2013).

Anti-forensic techniques to prevent memory analysis modify fragile signatures on a live system to block the path to evidence (Takahiro Haruyama, 2012). In addition, they can compromise the kernel data structure field, which has a semantic value, to make the memory analysis tools mislead the fields (Prakash et al.). Further, they can construct fake kernel objects to increase the analysis time (Jake Williams, 2014). Among these anti-forensic techniques, the one-byte abort factor showed extreme limitations in modern memory analysis algorithms by modifying only one-byte value used in memory analysis.

To deal with anti-forensic techniques for memory forensics, OS-Sommelier (Gu et al., 2012) and OS-Sommelier+ (Gu et al., 2015) suggest a code-based

* Corresponding author.
E-mail addresses: koungholee@gmail.com (K. Lee), hhu@nsr.re.kr (H. Hwang), kibom@nsr.re.kr (K. Kim), bbong@jnu.ac.kr (B. Noh).

http://dx.doi.org/10.1016/j.diin.2016.04.009

signature generation, which is very effective for OS fingerprinting on a system that is ensured integrity of the kernel code. Additionally, image-based signature (Roussev et al., 2014) is valuable for OS fingerprinting with a corpus of known kernel binaries. Furthermore, profile indexing (Cohen, 2015) identifies kernel versions using only the globally unique identifier (GUID) without known kernel binaries. However, the advance preparation of this method before comparing the profile indexes is long and is weak against interference. The Directory Table Base (DTB) identification of OS-Sommelier is excellent for the ×86 system but is complicated. Research about robust signatures for kernel data structures (Dolan-Gavitt et al., 2009; Lin et al.; Lin) needs to know the OS versions for accurate scanning. In the viewpoint of comprehensive analysis, these research works against anti-forensics handle only one element among the bootstrapping steps for memory analysis — OS fingerprinting, DTB identification, and obtaining kernel objects, which means that any collapse in the robust algorithms will lead to failure in the memory analysis.

Therefore, in this paper, we show the anti-forensic techniques and the limitations of modern analysis algorithms. Then, we suggest a novel memory analysis algorithm based on KiInitialPCR, which assures the bootstrapping analysis. It locates the KPCR structure, which has many robust fields, and uses the relationship among fields that point to global variables for OS fingerprinting. It also acquires DTB in the KPCR structure. In addition, it calculates the relative offset based on KiInitialPCR (instead of the kernel base) to list the kernel objects. Thus, we prove that it is an effective and comprehensive analysis algorithm that uses only one robust structure against anti-forensic techniques.

Section 2 explains how anti-forensics disturb memory analysis, and shows limitation of bootstrapping analysis used by existing tools. It also shows the limitations of current research against anti-forensic techniques. Section 3 evaluates the modern memory analysis methods, including the bootstrapping analysis against anti-forensics. Section 4 introduces the KiInitialPCR-based analysis to deal with anti-forensic techniques. This analysis carves the KiInitialPCR, identifies the OS version, and extracts the process list and the module list using non-modifiable fields in the memory. Section 5 shows the implementation of the suggested analysis procedure. Section 6 discusses the limitations of our system. The final section offers conclusions and directions for future research.

Background

Anti-forensics

Anti-forensics concentrates on how to make investigators fail to collect volatile evidence by modifying critical values used in memory analysis. An attacker with high privilege can modify kernel memory.

The one-byte abort factor attack (Takahiro Haruyama, 2012) shows that important signatures are easily overwritten by malware in such a way that memory analysis can fail to find it. In addition, semantic value manipulation (SVM) attack mutates semantic values, which are data values with important semantic meanings (Prakash et al.). Further, the attention-deficit-disorder (ADD) technique creates fake objects to lead investigators down a wrong path and increases the analysis time (Jake Williams, 2014).

Whereas the SVM and ADD can analyze physical memory images and extract volatile data irrespective if the data are genuine, modified, or fake, attacking bootstrapping analysis like the abort factor makes the analysis fail, which means that the investigator cannot collect any evidence from the physical memory image. Therefore, the attacking bootstrapping analysis is an important problem to be solved first.

Physical memory analysis must perform bootstrapping analysis, which is composed of OS fingerprinting meaning identifying the OS version, acquiring DTB, and obtaining the kernel data structures. Correct OS fingerprinting enables precise parsing of the kernel data structures with accurate structure layout. It also enables precise selection of the analysis algorithms, which are different in different versions. Acquisition of DTB enables reconstruction of the virtual address space, which is the mapping between the virtual and physical addresses. Obtaining the kernel data structures enables us to collect kernel data such as process and thread information.

A potential target of the attacking bootstrapping analysis is all modifiable memory, which does not cause noticeable differences such as crashes in system state with modification, used in the analysis.

Volatility (The Volatility Foundation, 2015), which is a famous memory forensic tool, uses KDDEBUGGER_DATA64 structure, which is known as KDBG, to identify the OS version with *Size* field and get the global kernel variables with fields named same as each variables like PsActiveProcessHead. Then, it uses the EPROCESS structure of the idle process to obtain the DTB with *DirectoryTableBase*.

Memoryze uses the EPROCESS structure of the system process to identify the OS version by matching DISPATCHER_HEADER signatures of all OS (e.g., \×03 \×00 \×1B \×00 and \×30 \×00 \×26 \×00) and by confirming whether the *ImageFileName* field is a "System" string or not. In addition, it obtains the DTB from the system process.

As mentioned in the abort factor, these well-known analysis algorithms use fragile signatures. The attacker can modify the "KDBG" string, "Idle" string or "System" string, which is critical memory values to the analysis, to abnormal value. As we show in Section 3, this technique still can disrupt memory analysis.

Rekall (The Rekall Team, 2015a) gathers the program database (PDB) information from GUID in the RSDS region of the kernel executables and identifies the OS version from the PDB information, which contains the structure layout information and global debugging symbols. It carves the RSDS region with "RSDS" signature and known PDB file names of the kernel executables (e.g., ntoskrnl.pdb). However, the region of the GUID and PDB filename is a modifiable memory.

Further, rekall uses the profile indexing method (Cohen, 2015) known as nt index to deal with the abort factor. In profile generation phase, rekall chooses arbitrarily 10–12 addresses among the virtual addresses of NOP (0×90) instructions preceding the function and of the string literals (e.g. "FILE_VERSION"). These addresses contain debugging

symbols in the kernel. Then, it generates profiles that are composed of the selected addresses and values contained in the address in each kernel version. In analysis phase, rekall obtains the DTB in the lead off. Then, it determines the exact kernel build version by comparing the values of the addresses in the profiles with the indexed values in the profiles. This comparison is called the profile-indexing method, and the compared virtual addresses are called "comparison points".

The profile indexing method suffers from some limitations. The comparison points are also modifiable despite of choosable one randomly. If one of them is modified, the analysis can fail or mislead kernel version. Moreover, it should reconstruct virtual address space for comparison and find kernel base with pe signatures (e.g. "This program cannot be run in DOS mode"). These pe signatures are easily modifiable without causing any system crashes.

Limitations against anti-forensics

OS fingerprinting

OS-Sommelier (Gu et al., 2012) generates signatures by hashing each redefined kernel page without pointer values. In addition, OS-Sommelier+ (Gu et al., 2015) removes the kernel module code in the kernel memory and leaves only the core kernel code on the kernel pages. Then, it hashes each kernel page to create signatures. This code-hashing-based approach might be prevented by anti-forensics owing to their complicated processes. Further, because it is proposed for virtual machine introspection in cloud system, it should ensure integrity of the kernel code. However, we cannot always assure the integrity of the kernel code in real world, especially in a live response situation.

The image-based identification of a kernel version (Roussev et al., 2014) generates signatures by hashing each page of the kernel executable binary and matching the kernel in a memory image with a corpus of known binaries by checking for similarity. However, we do not always have the actual binaries.

DTB identification

The DTB identification method suggested in the OS-Sommelier to reconstruct the virtual address space collects the DTBs such that at least one entry must exist that points to a valid target page directory table. Then, to select a valid DTB, it compares the similarities of the shared kernel memories among the collected DTBs. Because this criterion is based on the ratio of the number of kernel pages, it might identify invalid DTBs or fail to identify the DTBs if the ratio is changed.

Getting kernel structures

Robust signature research (Dolan-Gavitt et al., 2009) generates robust signatures for carving kernel data structures using non-modifiable fields, which crashes the system when they are modified. Siggraph (Lin et al.) and MACE (Lin) rely on point-to-relation or pointer constraints. Although these robust signature schemes are effective in carving kernel data structures, they should check the data structure signatures for various OS versions if it does not know the accurate structure layout without OS fingerprinting. In particular, the use of the point-to-relation relies on the precise identification of the DTB.

Memory analysis based on KPCR

The Windows kernel manages the processor information using the KPCR structure as shown in Fig. 1. The Kernel Processor Region Control Block (KPRCB), which is a substructure of the KPCR, contains the DTB in the *Cr3* field for virtual address translation.

The KPCR carving condition that is proposed to analyze the Windows XP memory (Zhang et al., 2009) cannot analyze Windows 7 because of the non-fixed virtual addresses of the KPCR and KPRCB. To solve this problem, the carving method using the fixed difference between the *Self* and *CurrentPrcb* fields in a 32-bit system is proposed (Shuhui et al., 2010). It is also applied to the 64-bit system in accordance with the difference between the *SelfPcr* and *Prcb* fields.

The KPCR-based analysis implemented for Windows 7 obtains instances of the EPROCESS structure from the *CurrentThread* field in the KPRCB structure and traverses a double-linked list to extract the process list. However, because obtaining invalid EPROCESS structures from the *CurrentThread* field could occur, the method of extracting the process list using the *PsActiveProcessHead* field which is the head entry of the process list in the KDBG structure is proposed (Thomas et al., 2013).

Assessments of anti-forensics

We developed a proof-of-concept driver that implements the anti-forensic techniques, and evaluated the modern tools and methods against this anti-forensic tool. Modification targets of the driver are described as follows:

- 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

```
nt!_KPCR
   …
   +0x01c SelfPcr          : Ptr32 _KPCR
   +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
      +0x3cc Number : Uint4B
      …
```

Fig. 1. KPCR structure layout in 32-bit system.

- Kernel PE signature: used to find kernel base for OS fingerprinting.
- Comparison points: used to identify the kernel build version, including the OS version.

We have modified the DISPATCHER_HEADER and the *ImageFileName* fields of the System and Idle processes and the *OwnerTag* and the *Size* fields of the KDBG structure according to how they are presented in the one-byte abort factor (Takahiro Haruyama, 2012). To evaluate the new analysis method, we also modified the RSDS region of the kernel executable, pe signatures of the kernel base, and the comparison points. These fields were modified to arbitrary values. All tested memory modifications do not cause any system crashes.

The evaluation system used is the Windows 7 SP1 64-bit with a fully updated inside VMware Workstation. We have tested a tool that implements KPCR-based analysis (Thomas et al., 2013) as well as the latest version of the volatility, rekall, and memoryze.

For the evaluation, we selected the extraction process list function as a common function that includes the steps of identifying the OS versions and obtaining the DTBs among the functions of the target tools.

The results show that all tested analysis tools result in analysis-fail state by at least one of the tested anti-forensic techniques, as listed in Table 1. It shows that not only do the analysis methods of the modern tools still have weaknesses to anti-forensics but also that an analysis method to deal with anti-forensics is easily disabled by modifying the other memories that are essential in the algorithm.

Even though the comparison points of the profile indexing are arbitrarily changeable, the profile indexing can possibly be defeated by anti-forensics, because it still uses a modifiable memory. In addition, it causes failure in finding kernel base if the pe signatures are modified.

The KPCR-based analysis method (Thomas et al., 2013) obtains only the DTB from the KPCR. When trying to extract the process list, it carves the KDBG and uses the *PsActiveProcessHead* field of the KDBG, which is a modifiable memory.

Memory analysis based on KiInitialPCR

Challenges

The modern memory analysis tools, as presented in Section 3, are still prone to be subverted by the attacking bootstrapping analysis. In addition, most state-of-the-art research against anti-forensics, as presented in Section 3, suffer from some limitations such as the need to ensure the integrity of the kernel code, collect known binaries, and run complicated algorithms. This means that any collapse in the three steps using the suggested robust algorithms leads to failure in the memory analysis. Therefore, we should ensure the bootstrapping analysis. Also, if possible, we should use only robust memory data, which causes noticeable differences in system state like BSOD or stopped state with modification, as mentioned in robust signature research (Dolan-Gavitt et al., 2009).

We need to find a satisfactory structure according to the following conditions.

- It should have the same structure layout across various versions, and we can apply the same carving rule to locate it.
- It should have robust fields that can be used for OS fingerprinting.
- It should have robust fields that include a DTB value.
- It should be a kernel global variable.

The reason for the kernel global variable in the conditions is for us to access other important kernel global variables by adding the relative offsets of that variable.

We could find a structure that satisfies the above conditions, which is called the KPCR structure. One of the KPCR structures is the kernel global variable named KiInitialPCR. Because the KPCR structure is a key data for managing and controlling the processor, the KiInitialPCR-based analysis method will be effective as long as Windows changes its OS architecture. Thus, we propose a novel robust algorithm that guarantees the bootstrapping analysis through KiInitialPCR.

Feature of KiInitialPCR

The number of KPCR structures is equal to the number of processors in the system because Windows kernel produces KPCR structures that manage each processor. The kernel beyond Windows 7 manages the first-generated KPCR structures as a kernel global variable KiInitialPCR. On the other hand, other KPCR structures are allocated in the dynamic memory area in the kernel.

We carve the KiInitialPCR, identify the OS version, and extract the process list from the KiInitialPCR. The reasons for the use of the KiInitialPCR are as follows:

First, the KPCR structure has almost the same fields in the Windows OSs and identical size with the system bit. We can approach the KPCR structures even if we do not know the OS version.

Table 1
Results of the analysis with anti-forensics. The ○ symbol indicates that the tool successfully extracts the process list from the physical memory image, and the × symbol indicates that the tool fails to analyze the image.

Memory modification target	Volatility 2.5	Memoryze 3.0	Rekall 1.4.1 (RSDS)	Rekall 1.4.1 (nt index)	KPCR
Idle process	×	○	×	○	○
System process	○	×	○	○	○
KDBG	×	○	○	○	×
RSDS	○	○	×	○	○
PE signatures	○	○	○	×	○
Comparison points	○	○	○	×	○

Second, the KiInitialPCR and other KPCR structures have many non-modifiable fields. Such fields are difficult for an attacker to modify without crashes and are good candidates for robust signatures.

In addition, the KiInitialPCR has a self-reference field, which means that it has a virtual address to itself, named as *SelfPcr* in the 32-bit system and *Self* in the 64-bit system. This self-reference field enables us to directly access other kernel global variables without finding the kernel base address if the distance is known in advance.

In this section, we generate the carving conditions with the KPCR fields where anti-forensics cannot be applied. This feature has not been dealt with in previous research called KPCR-based analysis method. Further, we carve the KiInitialPCR. Then we introduce the methods of OS fingerprinting and obtaining the kernel global variable PsActiveProcessHead from the KiInitialPCR.

Carving of KiInitialPCR

Generation of carving signature

The KPCR structure stores the processor information, and the kernel executes the code using the stored processor information. The system crashes when some processor information is changed to invalid values.

We generate robust carving signatures on the KPCR structure using the VMware Workstation because the KiInitialPCR has the same structure type as the other KPCRs. The robust signature research (Dolan-Gavitt et al., 2009) could allocate a new object to generate a carving signature. However, we cannot allocate an instance of a KPCR structure because it is only allocated at boot time under normal circumstances. Therefore, we simply reboot the system after each test is finished and let the system allocate new instances of KPCR structures.

We implement the KPCR carving tool using the KPCR carving method (Thomas et al., 2013) to locate the KPCR structure instance in the virtual memory file. It uses the condition where the difference between the *SelfPcr* and *Prcb* fields is 0120 in the 32-bit system and that between the *Self* and *CurrentPrcb* fields is 0180 in the 64-bit system. It also checks whether the physical address translated from the *Self* field is equal to the physical address of KPCR.

The environments that generate the signature are Windows 7 SP1, 8, 8.1, and 10 32/64-bit version that are fully updated and running on a virtual machine. Also it has the quad-core CPU to confirm whether the system crashes or not in case the KPCR structures are modified in the multiprocessor environment.

The fields used in generating the signature are used with the fields preceding the *CurrentPrcb* field in the 32-bit system and the *Prcb* field in the 64-bit system and with the same name, offset, and type size in the tested systems.

The robust fields among the chosen fields are the *SelfPcr*, *Prcb*, and *GDT* in the 32-bit version and the *Self*, *CurrentPrcb*, *GdtBase*, and *LockArray* in the 64-bit version. These fields in the 64-bit version perform the same role as 32-bit fields in sequence. In this paper, we call the field names as the fields of the 64-bit version, which has same meaning as each respective field in the 32-bit version.

Using the relationship between fields that causes the system crash during modification, as listed in Table 2, we generate the robust KPCR carving signatures that are non-modifiable. These signatures include the condition that uses the relationship between the *Self* and *CurrentPrcb* fields in the previous KPCR carving research. In the 64-bit system, the *LockArray* field has a fixed offset from the *CurrentPrcb* field to the *LockQueue* field of the KPRCB because it points to the *LockQueue* field.

A selection of KiInitialPCR

We should select the KiInitialPCR among the carved KPCRs because the carving condition locates not only the KiInitialPCR but also the other KPCR structures. The method of selecting the KiInitialPCR is to check whether the value of the *Number* field, which is always zero in a normal KiInitialPCR, is zero or not. The offset of the *Number* field is equal to 03 cc in the 32-bit system and 024 in the 64-bit system in all OSs. Similar to the previous KPCR-based analysis (Thomas et al., 2013), we check the validation of the KiInitialPCR by comparing the translated value of the *Self* field that self-referenced the carved offset.

Modification possibilities in carving

The KPCR carving signature is generated by the non-modifiable fields. In addition, the *Number* field, which is used to select the KiInitialPCR from the others, causes the system to crash in case it is modified. The *Cr3* field used to obtain the DTB is continuously renewed without a system error because the system only stores the CR3 register and does not use the register value stored in it. Therefore, it is not affected by the modification.

Memory analysis using KiInitialPCR

Identification of OS version

The *IdleThread* field of the KPRCB, which is a substructure of the KiInitialPCR, points to the kernel global variable KiInitialThread. On the other hand, this field in the other KPCR points to the ETHREAD structure which is allocated in the heap memory. As kernel global variables, KiInitialPCR and KiInitialThread are located at fixed locations from the kernel base in the same kernel build version, and the distance between these fields is fixed.

In a previous research (Cohen, 2015), the relative offsets, which are the distances from the kernel base, of the global kernel variables greatly vary with the kernel build version even in the same OS version. To check whether the distance between KiInitialThread and KiInitialPCR greatly varies or not, we gather the offsets of the kernel global variables from each PDB file corresponding to the kernel version.

The Windows system puts all updated files, including the kernel executables, in the WinSxS folder. The kernel executable contains a unique GUID that refers to the PDB file. Therefore, we installed Windows 7 SP1, Windows 8, Windows 8.1, and Windows 10 32/64 bit inside the VMware Workstation and updated each system from the initial state to a fully updated state. Subsequently, after gathering the kernel executables in the WinSxS folder in each system, we collected 139 PDB files from the public Microsoft symbol

Table 2
KPCR CARVING CONDITIONS generated by non-modifiable fields. && indicates the Boolean operator "AND". % indicates Mod operation. & indicates the bitwise operator "AND". The 32-bit KPCR has no field corresponding to the *LockArray* field of the 64 bit. Union means a common condition of the robust fields.

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

server with Debug Interface Access (DIA) interface (Debug interface, 2015).

The rekall provides public profiles repository that has the GUIDs of PDB files (The Rekall Team, 2015b). But these profiles don't contain any major version or kernel version information. This means, although we collect some kernel global variable's offsets using GUID in these profiles, we cannot know an exact version corresponding with the GUID. Thus, we collected PDB files only in a virtual environment. As mentioned in (Cohen, 2015), there are many more versions of the kernel in the real world than our collection. We show continual patterns of global variable offsets to assume that the other kernel version has a similar pattern with these patterns.

Fig. 2 shows the distances that we collected from the KiInitialPCR to the KiInitialThread. The offsets are constant or contain little changes in the viewpoint of each OS version. In the Windows 7 SP1 version, for example, the offset has a constant value of 38,784 in the 32-bit non-PAE, 38720 in the 32-bit PAE, and 57,280 in the 64 bit. It means that any global variable between KiInitialPCR and KiInitialThread across various versions in the same major version is not added or not removed. Furthermore, the distances between these two variables do not overlap on each OS version.

Therefore, we identify the OS version using the difference in the *Self* and *IdleThread* fields of the KiInitialPCR. We use only a few differences to identify the OS version because we know paging mechanism (PAE, non-PAE) and machine bit in the previous steps. We call these differences as version signatures.

Extraction of process list

Extraction of the process list is a main and common function among modern memory forensic tools because the EPROCESS's instance contains and points to a number of

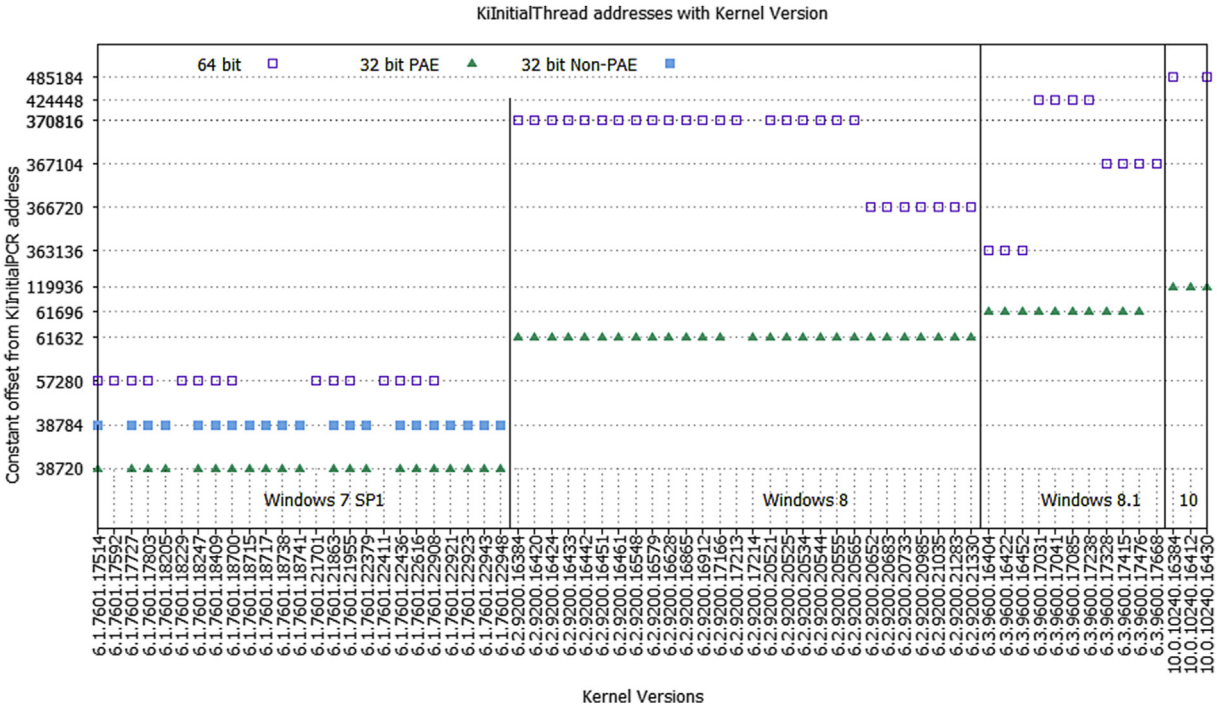

Fig. 2. Offsets between the KiInitialPCR and KiInitialThread in various kernel versions. These offsets were collected from the PDB files. Because Windows 8, 8.1, and 10 only run on processors that support PAE, no non-PAE(I386) offset is present.

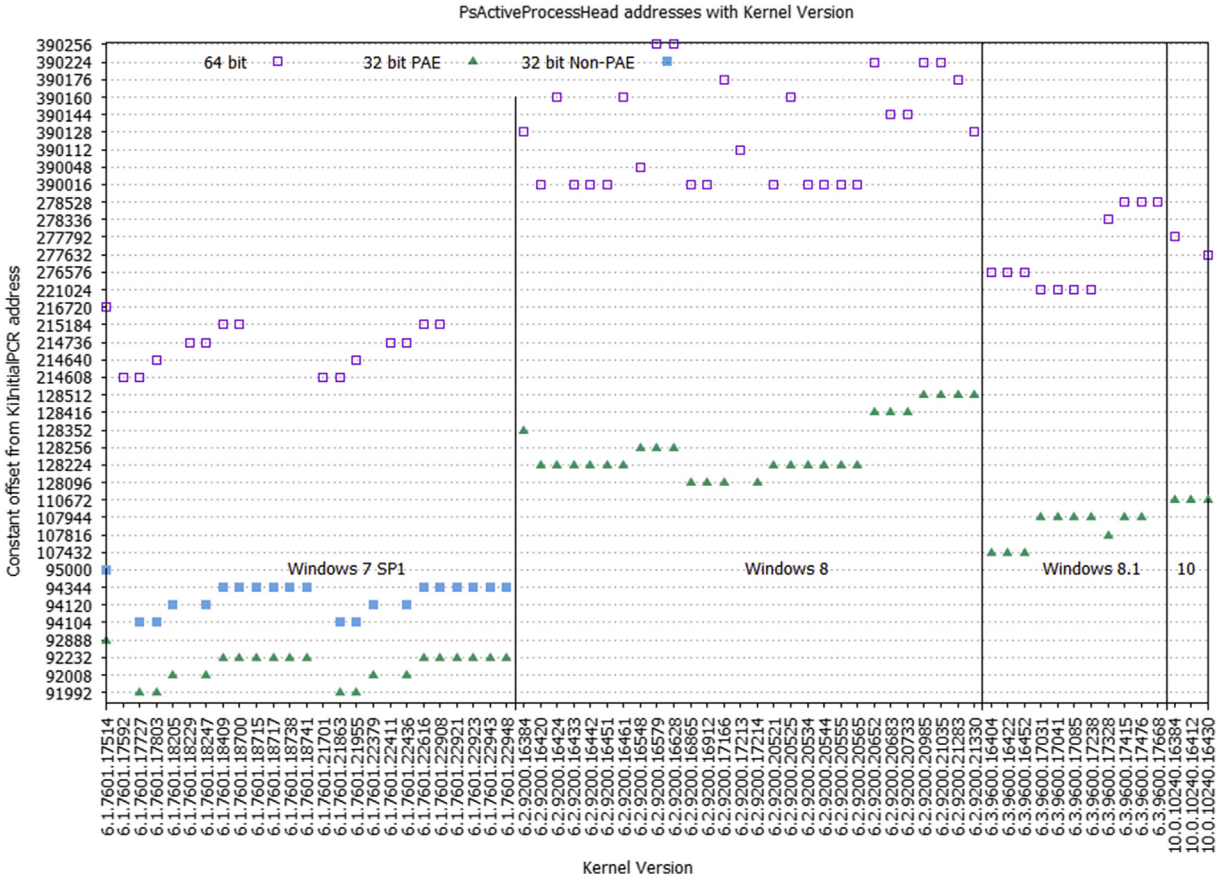

Fig. 3. Offsets between the KiInitialPCR and PsActiveProcessHead in various kernel versions. These offsets were collected from the PDB files. Because Windows 8, 8.1, and 10 only run on processors that support PAE, no non-PAE (I386) offset is present.

other related data such as thread, module and object table information. Therefore, we show how to extract active process list from the KiInitialPCR.

We collected the distances from the KiInitialPCR to the PsActiveProcessHead from the collected PDB files, as shown in Fig. 3. In the differences, the PsActiveProcessHead offsets vary more than the KiInitialThread, especially in the 64-bit system than in the 32-bit system. However, from the viewpoint of the OS version, an offset tends to repeat older offsets. For example, four offsets (214,608, 214,640, 214,736, and 215,184) are sequentially repeated on the 64-bit Windows 7 SP1.

Unfortunately, no direct method is available to determine a valid PsActiveProcessHead without an exact kernel build version. Therefore, we need to validate these offsets. To reduce the number of offsets to validate, we made finite sets of the offsets based on each version signature. The cardinalities of the sets are fewer than eight, which may add new members with the kernel update. We only use one set among them in the analysis because we know the version signature.

We need to validate each candidate of the finite set. We check whether the list entry completely traverses or not and validate each EPROCESS structure of the traversable links. The EPROCESS structure validity check uses robust signatures for EPROCESS (Dolan-Gavitt et al., 2009) for each link. After these two validations are successfully finished, we obtain the valid PsActiveProcessHead and extract the process list. Then, we can extract the threads, handles, loaded modules, and other volatile data that exist in the process memory from the extracted active processes. Also, we can find a hidden process by comparing the listed processes with the carved processes.

We also obtain the kernel modules by listing entries of the PsLoadedModuleList. We obtain the offsets between PsLoadedModuleList and KiInitialPCR. Similar to the PsActiveProcessHead offsets, we develop finite sets of the offsets based on each PsActiveProcessHead, instead of the version signature. The cardinalities of the sets are fewer than three. In our experiment, we know that the *InLoadOrderLinks* field of the LDR_DATA_TABLE_ENTRY, which is the structure of the kernel module, is the robust field. After we check whether the list entry completely traverses or not, we obtain the valid PsLoadedModuleList and extract the module list.

Modification possibilities during the analysis process

This section is composed of two parts: identifying the OS version and validating the offset of the PsActiveProcessHead and PsLoadedModuleList from KiInitialPCR.

When the *Self* and the *IdleThread* fields are modified, we find that it immediately causes system crash. In addition, the genuine PsActiveProcessHead causes system crash when it is modified. Some attackers may make a spurious list composed of fake objects in the offsets of the set, such as the ADD attack. However, the memory region of the offsets is still used in the kernel, which means that the attacker cannot easily manipulate the memory at the offsets for the concerned system crash. In addition, even if the attacker succeeds in manipulating the memory of the offsets, we can check only our offsets up to a maximum of eight, not entire memory. Also the validation of each entry cannot be subverted because we check the robust carving condition of the EPROCESS structure fields for each link including the head.

System implementation

An analysis system based on KiInitialPCR, as shown in Fig. 4, is divided into the database management of offsets and memory analysis using the offsets. A Database management is implemented with cpp language. Also, memory analysis is implemented with python to use the volatility framework.

Whenever the Windows kernel version in the virtual machine is updated, the database management acquires a PDB information and stores the KiInitialThread and PsActiveProcessHead offsets from the KiInitialPCR into the global variable offset database.

The memory analysis obtains the latest offset information, which requests it from the database, identifies the OS version, and extracts the process list.

The memory analysis procedure shown in Fig. 5 consists of the procedures of carving the KiInitialPCR, identifying the major OS version, and finding the PsActiveProcessHead needed to extract the process list. We modified partial OS fingerprinting code of volatility.

The memory analysis obtains latest offset information which requests it to the database, and identifies the operating system version and extracts the process list.

Our system carves the KiInitialPCR by checking every 0×100 bytes, which is memory alignment of the KPCR structure. After the carving, the system determines machine bits due to the difference between the 32- and 64-bit carving conditions. Because non-PAE and PAE, which are virtual address translation mechanisms, are used in the 32-bit system, the system checks both mechanisms, when validating the carved KiInitialPCR by *Cr3* field.

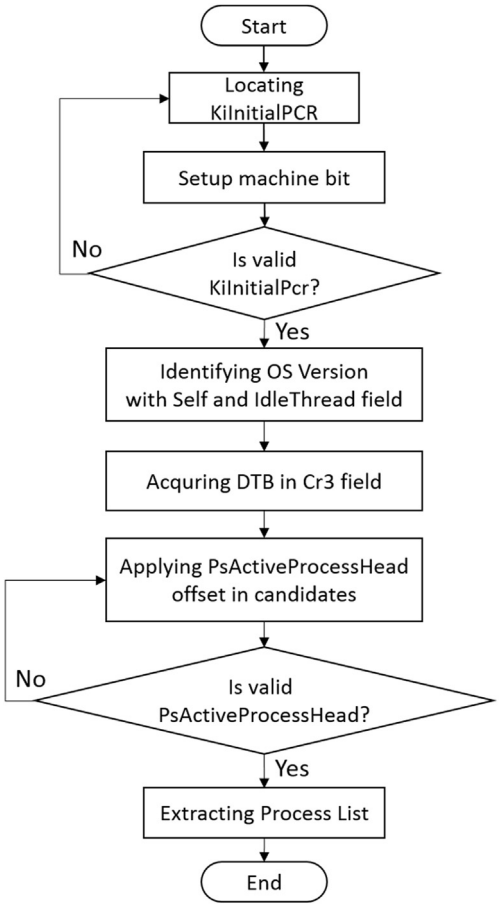

Fig. 5. Flowchart interpreting the steps for extracting the process list.

Before the OS fingerprinting, the system learns the machine bit and the virtual address translation mechanism. Therefore, we can exactly identify the OS version by comparing the identification key belonging to this known information.

The system uses the DTB value stored in the *Cr3* field of the KiInitialPCR for translating each link entry starting from the PsActiveProcessHead to the physical offset of the memory image. Then, it extracts the process list.

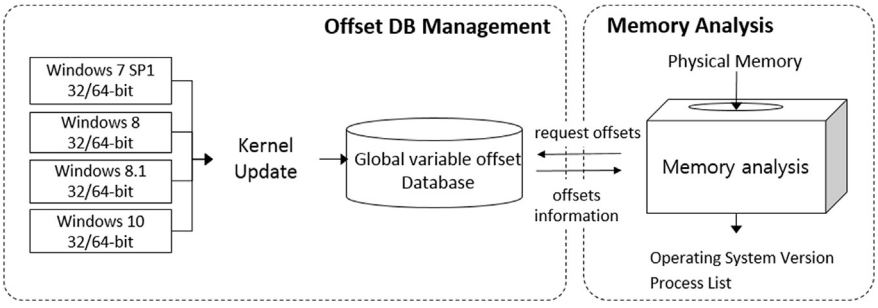

Fig. 4. System implementation. The global variable offset database is updated when new updates are released.

We collected memory images from 20 real machines, then we measure performance of our system. In case of Windows 7 images, average analysis time is 2 s, due to KiInitialPcr is located at low physical address. Also, images of Windows 8.1 and Windows 10 take 4 min for 8 GB, due to the KiInitialPCR is located at the end of file. In contrast, the kpcr plugin of volatility takes longer than over 1 h to the same memory images, because it checks only address equality between SelfPcr and the physical offset on every byte. Our higher speed comes from value-based signatures and checking 0×100 byte unit.

Discussion

In this paper, we have proven that we can analyze the memory using only one robust structure. However, our system suffers from some limitations as outlined in the following.

Lack of information

Our system requires known GUIDs and OS versions corresponding to each GUID to analyze the physical memory. Therefore, the main problem of our system is the lack of information on the GUIDs and OS versions. If we only know the GUID without the OS version, we can use the PDB file provided by the Microsoft public server. Although the PDB file does not contain the version information, we can approximately learn the version based on the existing symbols and field names or the field offset of a certain structure that exists in the profile.

Relocating kernel objects

Our system identifies the OS versions using *Self* field and *IdleThread* field of the KPCR structure. The attacker would adjust the gap of the fields to abort OS fingerprinting by relocating the KiInitialPCR and KiInitialThread to other allocated memory. We implemented it as simple driver and tested it.

First, we copied the KiInitialPCR and make the *Self* field point the copied one. Since the processor state is different before and after the copy, the system has stopped at once. Second, we copied the KiInitialThread and make the *IdleThread* field point the copied one. It generates the system crashes after a few minutes on Windows 8, 8.1 and 10. On the other hand, the system looks like normal on Windows 7.

All PDB files we have gathered indicate that the remainder after division of the KiInitialPCR offset by 0×1000 is 0×d00 or 0×c00 on Windows 7 and 0×1000 on Windows 8, 8.1 and 10 (e.g. 0×82f3ed00 on Windows 7 and 0×8182b000 on Windows 10). Therefore, if the result of *Self* field mod 0×1000 is non-zero, we can consider a version of memory image as Windows 7 without *IdleThread* field. If not, we checkout difference between *Self* field and *IdleThread* field of the KPCR structure which is an instance of KiInitialPCR. Thus, the attacker cannot easily defeat our system by relocating important fields of the system.

Other anti-forensic techniques

An attacker can make spurious KiInitialPCRs to increase the analysis time. In this case, because our automated system can gather all the KiInitialPCRs that satisfy the condition on the entire memory image, the analyst has no option but to rely on his common sense to classify whether they are genuine or fake. However, our system cannot deal with SVM attacks.

Collecting other data structures

Because of the characteristics of the robust signature research, we can only validate certain structures accessed and read by the kernel or those with sufficient pointer fields to achieve pointer relationship signatures.

Conclusion

In this paper, we have proposed a KiInitialPCR-based physical memory analysis methodology using non-modifiable fields. Whereas most published research works against anti-forensics handle only one among the OS fingerprinting, DTB identification, and obtaining the kernel object steps, we analyze one structure that has a number of robust fields to identify the OS version and acquire the DTB.

The contributions of our work in this field are summarized:

- We guarantee the bootstrapping analysis, and they are not subverted by anti-forensic techniques.
- Our OS fingerprinting and DTB identification parts allow effective application of the robust carving signatures relying on correct operating system information.
- Our robust kernel object listing can find hidden objects by comparing them with carved objects.

In the future, we hope to identify exact kernel versions using only the robust fields. Further, we will deal with the SVM attack and the ADD technique.

References

Betz C. Memparser. 2005. http://www.dfrws.org/2005/challenge/memparser.shtml.

Cohen MI. Characterization of the windows kernel version variability for accurate memory analysis. Digit Investig 2015;12:S38–49. http://linkinghub.elsevier.com/retrieve/pii/S1742287615000109.

Debug interface access sdk, https://msdn.microsoft.com/en-us/library/x93ctkx8.aspx (2015).

Dfrws 2005 forensics challenge, http://www.dfrws.org/2005/challenge/ (2005).

Dolan-Gavitt B, Srivastava A, Traynor P, Giffin J. Robust signatures for kernel data structures. In: ACM conference on computer and communications security (CCS); 2009. p. 566.

Gu Y, Fu Y, Prakash A, Lin Z, Yin H. OS-SOMMELIER: memory-only operating system fingerprinting in the cloud. In: ACM symposium on cloud computing (SoCC); 2012. p. 5.

Gu Y, Fu Y, Prakash A, Lin Z, Yin H. Multi-Aspect, Robust, Mem 2015;2(4):380–94.

Harris R. Arriving at an anti-forensics consensus: examining how to define and control the anti-forensics problem. Digit Investig 2006;3:44–9 (SUPPL.).

Jake Williams AT. Add – complicating memory forensics through memory disarray. 2014. https://archive.org/details/ShmooCon2014-ADD-Complicating-Memory-Forensics-Through-Memory-Disarray.

Z. Lin, MACE: high-coverage and robust memory analysis for commodity operating systems.

Z. Lin, J. Rhee, X. Zhang, D. Xu, X. Jiang, SigGraph: brute force scanning of kernel data structure instances using graph-based signatures, Proc. of 18th annual network & distributed system security symposium.

Mrdovic S, Huseinovic A. Forensic analysis of encrypted volumes using hibernation file. In: 2011 19th telecommunications forum, TELFOR 2011-Proceedings of papers; 2011. p. 1277–80.

Olajide F, Savage N, Akmayeva G, Shoniregun C. Digital forensic research – the analysis of user input on volatile memory of windows application. In: IEEE world congress on internet security (WorldCIS); 2012. p. 231–8.

A. Prakash, E. Venkataramani, H. Yin, Z. Lin, Manipulating semantic values in kernel data structures: attack assessments and implications, Proceedings of the international conference on dependable systems and networks.

Roussev V, Ahmed I, Sires T. Image-based kernel fingerprinting. Digit Investig 2014;11:S13–21. http://linkinghub.elsevier.com/retrieve/pii/S1742287614000565.

Shuhui Z, Lianhai W, Ruichao Z, Qiuxiang G. Exploratory study on memory analysis of Windows 7 operating system. In: ICACTE 2010-2010 3rd International Conference on Advanced Computer Theory and Engineering, vol. 6; 2010. p. 373–7. Proceedings.

Stüttgen J, Cohen M. Anti-forensic resilient memory acquisition. Digit Investig 2013;10:105–15 (SUPPL.).

Takahiro Haruyama HS. One-byte modification for breaking memory forensic analysis. 2012. https://media.blackhat.com/bh-eu-12/Haruyama/bh-eu-12-Haruyama-Memory_Forensic-Slides.pdf.

The Rekall Team. The rekall memory forensic framework. 2015. http://www.rekall-forensic.com/.

The Rekall Team. The rekall profile repository. 2015. https://github.com/google/rekall-profiles.

The Volatility Foundation. Volatility foundation. 2015. http://www.volatilityfoundation.org/.

Thomas S, Sherly KK, Dija S. Extraction of memory forensic artifacts from windows 7 RAM image. In: 2013 IEEE conference on information and communication technologies, ICT 2013 (ict); 2013. p. 937–42.

Zhang R, Wang L, Zhang S. Windows memory analysis based on KPCR. In: 5th international conference on information assurance and security, IAS 2009vol. 2; 2009. p. 677–80.