Süleyman Demirel Üniversitesi
Fen Bilimleri Enstitüsü Dergisi
Cilt 23, Sayı 2, 498-504, 2019

Süleyman Demirel University
Journal of Natural and Applied Sciences
Volume 23, Issue 2, 498-504, 2019

DOI: 10.19113/sdufenbed.529039

Development of Kernel Mode RAM Driver for RAM Image on Windows

Ahmet Ali SÜZEN*[1], Kubilay TAŞDELEN[2], Ecir Uğur KÜÇÜKSİLLE[3]

[1]Isparta Uygulamalı Bilimler Üniversitesi, Uluborlu Meslek Yüksekokulu, Bilgisayar Teknolojileri Bölümü, 32100, Isparta, Türkiye
[2]Isparta Uygulamalı Bilimler Üniversitesi, Teknoloji Fakültesi, Elektrik Elektronik Mühendisliği Bölümü, 32100, Isparta, Türkiye
[3]Süleyman Demirel Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, 32100, Isparta, Türkiye

(Alınış / Received: 19.02.2019, Kabul / Accepted: 19.04.2019, Online Yayınlanma / Published Online: 30.08.2019)

Keywords
C++,
Kernel-mode driver,
Memory forensics,
RAM architecture

Abstract: In the field of computer forensics live analysis through immediate intervention is an important way of gathering electronic evidence. The way to obtain evidence from volatile data using live analysis is to take an image of the RAM (Random Access Memory). The entire RAM has to be copied in order to import data from this image. However, since the user mode is the default mode in Windows operating systems only the running processes can be accessed. Therefore, RAM imaging software needs to work at Kernel Mode level. In this study, a RAM driver was developed using WDK (Window Driver Kit) to enable RAM imaging software to run in Kernel Mode. The developed driver works on Windows 8, 8.1 and 10 (32 bit and 64 bit) operating systems. Virtual addresses, physical addresses and table pages for RAM can be accessed using the developed RAM driver. In this way, image acquisition software using this driver is able to carry out bit-to-bit copying of RAM. In addition, a program to import a RAM image in c ++ using this driver has also been developed. When the image retrieval software is installed in RAM it occupies a meager 156 KB of space. Compared to the existing image acquisition software, the developed RAM driver and software seem to use the least RAM. In addition, there are no examples of Kernel Mode RAM Drivers developed using WDK in the literature.

Windows'da RAM İmajı için Kernel Mode RAM Sürücüsü

Anahtar Kelimeler
C++,
Çekirdek mod sürücü,
Hafıza adli bilişimi,
RAM mimarisi

Özet: Adli bilişim alanındaki elektronik delil etme sürecinde, ilk müdahale ile canlı analiz önemli bir yer tutmaktadır. Canlı analiz ile uçucu verilerden delil elde etme, RAM (Random Access Memory) 'in imajı alınarak gerçekleştirilir. Alınan imajdan veri kazımak için RAM' in tamamının kopyalanması gerekmektedir. Fakat Windows işletim sisteminde default olarak User-Mode kullanıldığı için sadece çalışan process'lere erişilebilmektedir. Bu nedenle RAM imajı yazılımlarının Kernel-Mode seviyesinde çalışması gerekmektedir. Bu çalışmada, RAM imajı yazılımlarının Kernel-Mode'da çalışabilmesi için WDK (Window Driver Kit) ile RAM sürücüsü geliştirilmiştir. Geliştirilen sürücü, Windows 8, 8.1 ve 10 (32 bit ve 64 bit) işletim sistemlerinde çalışmaktadır. Geliştirilen RAM sürücü aracılığıyla RAM'in sanal adreslerine, fiziksel adreslerine ve tablo sayfalarına erişilebilmektedir. Böylece sürücüyü kullanan imaj alma yazılımların, RAM'i bit-to-bit kopyalamasına imkân sağlanmaktadır. Ayrıca, bu sürücü kullanarak c++ dilinde bir ram imajı alma programı geliştirilmiştir. İmaj alma yazılımı RAM'e yüklendiğinde 156 KB'lık yer kaplamaktadır. Geliştirilen RAM sürücüsü ve yazılımının, imaj alma yazılımları arasında RAM'ı en az kullandığı görülmektedir. Ayrıca literatürde WDK ile geliştirilen Kernel Mode RAM sürücüsü hakkında çalışma bulunmamaktadır.

**Corresponding author: ahmetsuzen@isparta.edu.tr*

1. Introduction

In the prosecution of cybercrimes, electronic evidence is needed rather than physical evidence. In particular, RAM must be tampered with in order to collect evidence [1]. This means copying the data stored in the RAM onto a disc using image acquisition software. Electronic evidence is obtained by applying the data-carving methods noted in image [2].

Studies conducted in the 1990s revealed just how important the data stored in RAM is. However, no method for obtaining this data had yet been developed [3]. The first ever RAM image acquisition and analysis application is known as KNTTools, and was developed in 2005. Using KNTTools, the RAM image is subjected to a thread search analysis [4]. In studies made between 2006 and 2012 using Windows XP data of less than 4KB was extracted from RAM using search and carve methods. With the restriction of full access to RAM in the Windows Vista operating system, these studies lost their validity. [5-10]. For Windows Vista, 7, 8, 8.1 and 10 operating systems a Kernel Mode RAM driver is required in order to obtain RAM images [11]. RAM driver and image acquisition software using this driver have been developed by commercial companies and open source developers. However, the studies in the literature were all carried out using commercially available software [12].

In Windows operating systems the RAM image shows what operations were performed by the user. In studies in the literature RAM was scanned to identify pictures, document files, malware detection and running processes [13-16]. The results obtained from the scans were found to vary depending on the operating system version, RAM capacity and received image management. In the analysis process, high success has been achieved in the data resulted in the detection of images, malware and running processes [13, 14]. However, it is seen that the success rate for obtaining the data from document files is also very low [15, 16].

There are two types of security level in the Windows operating system: ring0 (kernel mode) and ring3 (user mode). The commands sent directly from the processor to the RAM are processed in Kernel Mode. Applications that run through APIs run in User Mode since they do not have direct hardware access [17].

In Windows operating systems 50% of the available RAM is allocated to Kernel Mode for exclusive use and the remaining 50% to User Mode [18]. The operating system makes virtual addressing in RAM with 4KB page sizes. For a 32-bit operation, the maximum virtual address space is 2GB in size and in the address range 0x00000000 through 0x7FFFFFFF. For a 64-bit operation the maximum virtual address space is 8 TB in size in the address range 0x000'00000000 - 0x7FF'FFFFFFFF [19].

In our study a Kernel Mode RAM Driver was developed for use in RAM image acquisition software. When the driver is installed on the system it provides Kernel Mode access to the software using it. This allows access to all of the virtual and physical addresses created for RAM. The RAM image acquisition software was developed using the coded Kernel Mode RAM Driver.

2. Methods

2.1. Kernel mode and user mode

There are four different security levels for x86 or x64 processors. These are ring0 (kernel mode), ring1, ring2 and ring3 (user mode) [20]. In Windows operating systems, Kernel and User Mode are utilized. The processor switches between the two modes depending on the type of command that runs on it. Applications run in User Mode. The core operating system components also run in Kernel Mode. At the same time, while many drivers run in Kernel Mode some drivers are able to run in User Mode. When a User Mode application is started, the Windows operating system creates a process for the application. The process provides an application-specific virtual address space and a custom handling table. Since an application's virtual address space is private the application cannot change the data of another application. Each application runs on its own. If a crash happens the lockdown is limited to this application. Other applications and operating systems are not affected by the lockdown [21].

At the User Mode security level the virtual address space is limited. A process running in User Mode cannot access the virtual addresses reserved for the operating system. Limiting the virtual address space of a User Mode application prevents the application from changing and damaging critical system data. Commands running in kernel mode share a single common virtual address space. Therefore, if the Kernel Mode driver mistakenly writes to a different virtual address it could put the operating system or other driver's data in jeopardy. In addition, if a Kernel Mode driver generates an error in the system the operating system is also affected by this error [22]. The diagram in Figure 1 shows the interaction structure between User and Kernel Mode components.

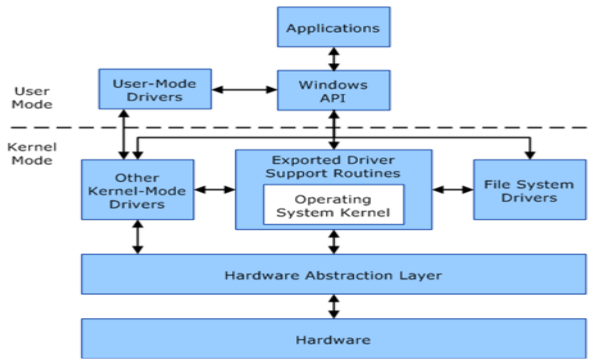


Figure 1. User and kernel mode to interaction [23]

2.2. RAM management

The virtual address range available for a process is called the process's virtual address space. In User Mode each process has its own virtual address space. For a 32-bit process the virtual address space is between 0x00000000 and 0x7FFFFFFF. For a 64-bit process the virtual address space is between 0x000'00000000 - 0x7FF'FFFFFFFF. The virtual addresses defined for the process are also called virtual memory [18].

Figure 2 shows the location of two 64-bit processes named myapp.exe and myapp1.exe in RAM. Both processes are located at addresses in the virtual address space. Both processes are stored in shadowed 4 KB pages. In the virtual space three adjacent addresses of the MyApp.exe process and the two adjacent addresses in the myapp1.exe process are mapped to non-adjacent addresses on the RAM page. Again, the two processes are also paged in an address that is different from the addresses in the virtual address space [24].

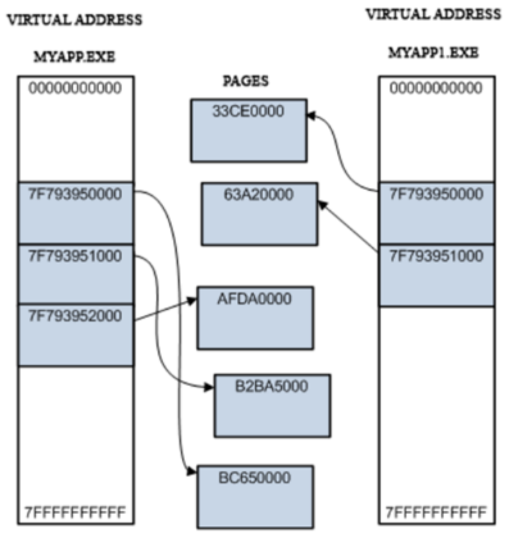


Figure 2. Samples process management.

In User Mode the virtual address space for the 32-bit operating system is 2 GB and 8 TB for the 64-bit system. In Kernel Mode the virtual address space for the 32-bit operating system is 2 GB and 248 TB for the 64-bit system [24].

2.2.1. EPROCESS

All processes running on the Windows operating system are kept in the EPROCESS (Executive Process) table. The EPROCESS table stores the process ID, creation date, release date, and exit status information of the running process [19].

2.2.2. PEB

The Process Environment Block (PEB) is where one of the data structures in EPROCESS, the properties, attributes, memory addresses, operating system version and DLL information of the running process are stored. In addition, the initial address of the running process is also accessed from the PEB [12].

2.2.3. File_Object

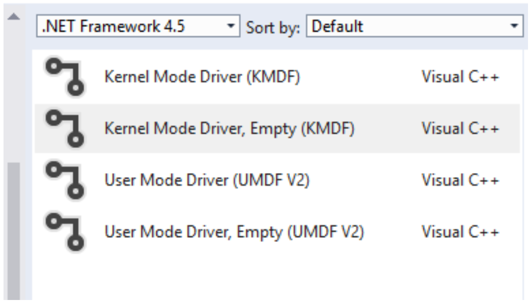
The File_Object table is where I / O functions, file names, and cache information of the terminated process created by the Windows operating system are kept. At the same time, File_Object also contains the folder and device information for the open files [25].

2.2.4. Pagefile.sys

Pagefile.sys is the system file that the Windows operating system uses as a temporary memory when RAM capacity is insufficient. This system file must be included in the RAM image in order to perform a complete analysis in computer forensics processes [25].

2.3. Windows driver kit

The WDK is the library used to develop User or Kernel Driver Mode with C ++ using the Visual Studio platform as shown in Figure 3. To use a User or Kernel Mode Driver developed with the WDK it has to be signed and tested. The Windows Hardware Certification Kit and the Hardware Lab Kit are used for signing and testing in the WDK.



.NET Framework 4.5 Sort by: Default

Kernel Mode Driver (KMDF)	Visual C++
Kernel Mode Driver, Empty (KMDF)	Visual C++
User Mode Driver (UMDF V2)	Visual C++
User Mode Driver, Empty (UMDF V2)	Visual C++

Figure 3. Varieties of WDK driver.

3. The Study

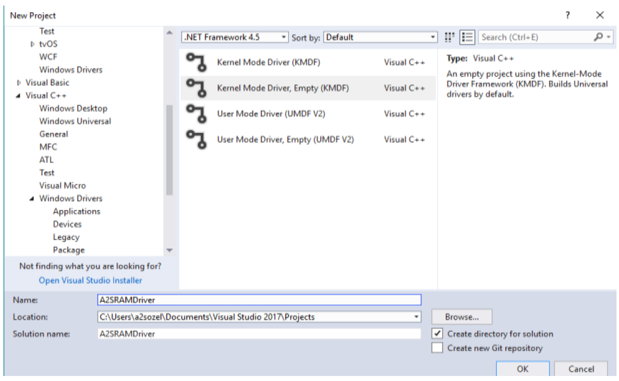
3.1. Kernel mode ram driver development

The Windows driver was developed with the C ++ programming language in the WDK template using the Windows Driver Frameworks (WDF) library. The WDK required to develop the driver is included in the Visual Studio 2015 and 2017 platforms. The WDK is also used in Windows 7 and later operating systems.

The developed driver needs to have a digital signature in order to be able to run in the operating system. Extended Validation Code Signing Certificates (EVCSC) were used to provide digital signatures for the Kernel Mode Driver. EVCSC are digital signature certificates that include the hardware security modules required for the operation of the Kernel Mode driver in operating systems. This certificate is purchased from security companies for commercial use. Digital signing is done with the Test Certificate Kit during driver development and testing with WDK. The Test Certificate Kit is software that enables the Kernel Mode driver to be tested and used in the Windows operating system. To use the certificate generated with the Test Certificate Kit the Windows operating system must be configured to "disable driver signing enforcement" in the initial settings.

When a new kernel driver is installed in the operating system it communicates with I/O, Power and Plug & Play managers. During the initial installation of the driver a request is sent to the I/O manager. This request is made with the IRPs (I / O Request Packets) parameters in the driver file. The driver to be loaded must respond correctly to the I/O manager's request with the IRPs parameter. If incorrect parameters are sent this causes a lockout, freezing or blue screen errors since it will be through Kernel Mode [12].

The driver development process begins with the opening of a new Kernel Mode Driver template via Visual Studio 2017, as shown in Figure 4.



.NET Framework 4.5 Sort by: Default

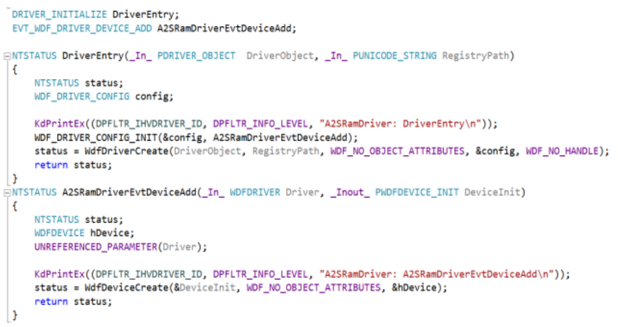
Kernel Mode Driver (KMDF) Visual C++
Kernel Mode Driver, Empty (KMDF) Visual C++
User Mode Driver (UMDF V2) Visual C++
User Mode Driver, Empty (UMDF V2) Visual C++

Type: Visual C++
An empty project using the Kernel-Mode Driver Framework (KMDF). Builds Universal drivers by default.

Name: A2SRAMDriver
Location: C:\Users\a2sozel\Documents\Visual Studio 2017\Projects
Solution name: A2SRAMDriver
[✓] Create directory for solution
[ ] Create new Git repository

Figure 4. Template of kernel mode driver.

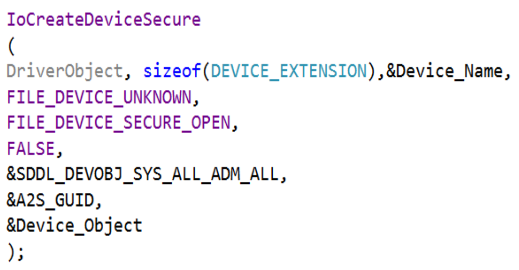
In the new project template, driver.c has been created in order to create the driver, and the c ++ library file address_pte_list has been created in order to access the addresses on the PTE. The functions DriverEntry and A2SRamDriverEvtDeviceAdd, which send requests to the I/O manager in the Driver.c file are defined as in Figure 5.



```
DRIVER_INITIALIZE DriverEntry;
EVT_WDF_DRIVER_DEVICE_ADD A2SRamDriverEvtDeviceAdd;

NTSTATUS DriverEntry(_In_ PDRIVER_OBJECT  DriverObject, _In_ PUNICODE_STRING RegistryPath)
{
    NTSTATUS status;
    WDF_DRIVER_CONFIG config;

    KdPrintEx((DPFLTR_IHVDRIVER_ID, DPFLTR_INFO_LEVEL, "A2SRamDriver: DriverEntry\n"));
    WDF_DRIVER_CONFIG_INIT(&config, A2SRamDriverEvtDeviceAdd);
    status = WdfDriverCreate(DriverObject, RegistryPath, WDF_NO_OBJECT_ATTRIBUTES, &config, WDF_NO_HANDLE);
    return status;
}
NTSTATUS A2SRamDriverEvtDeviceAdd(_In_ WDFDRIVER Driver, _Inout_ PWDFDEVICE_INIT DeviceInit)
{
    NTSTATUS status;
    WDFDEVICE hDevice;
    UNREFERENCED_PARAMETER(Driver);

    KdPrintEx((DPFLTR_IHVDRIVER_ID, DPFLTR_INFO_LEVEL, "A2SRamDriver: A2SRamDriverEvtDeviceAdd\n"));
    status = WdfDeviceCreate(&DeviceInit, WDF_NO_OBJECT_ATTRIBUTES, &hDevice);
    return status;
}
```

Figure 5. Use of DriverEntry and A2SRamDriver EvtDeviceAdd functions.

The FILE_READ_DATA function of the library wdm.h in WDF is used to give the driver access to RAM addresses. The IoCreateDeviceSecure function needs to be set as shown in Figure 6 to give the driver access to RAM.



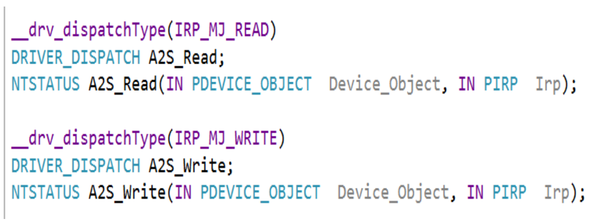
```
IoCreateDeviceSecure
(
DriverObject, sizeof(DEVICE_EXTENSION),&Device_Name,
FILE_DEVICE_UNKNOWN,
FILE_DEVICE_SECURE_OPEN,
FALSE,
&SDDL_DEVOBJ_SYS_ALL_ADM_ALL,
&A2S_GUID,
&Device_Object
);
```

Figure 6. Call of IoCreateDeviceSecure functions.

After the driver is loaded into the system, device identification is performed to access RAM via the image acquisition software.

#define A2S_DEVICE_NAME L"a2sram"

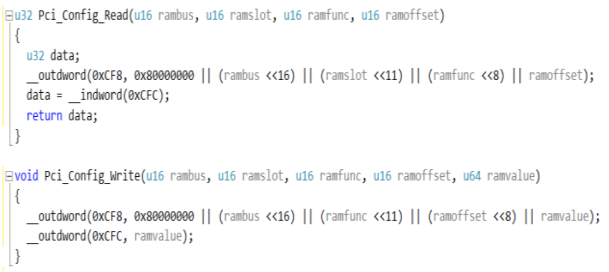
When the driver is loaded into the system, the IRPs parameters are communicated to the I/O manager via the IRP_MJ_READ and IRP_MJ_WRITE functions, which are referenced from the library wdm.h as shown in Figure 7.



```
__drv_dispatchType(IRP_MJ_READ)
DRIVER_DISPATCH A2S_Read;
NTSTATUS A2S_Read(IN PDEVICE_OBJECT  Device_Object, IN PIRP  Irp);

__drv_dispatchType(IRP_MJ_WRITE)
DRIVER_DISPATCH A2S_Write;
NTSTATUS A2S_Write(IN PDEVICE_OBJECT  Device_Object, IN PIRP  Irp);
```

Figure 7. IRP_MJ_READ and IRP_MJ_WRITE functions.

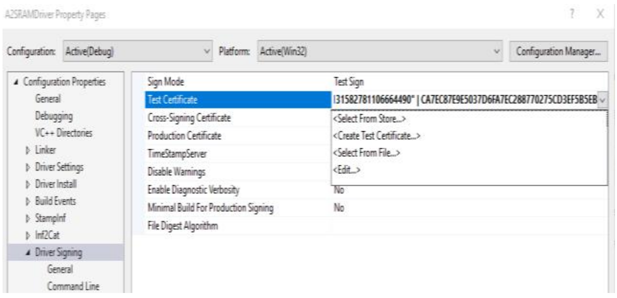
The __outword and _inword functions of the wdm.h library provide access to RAM address ranges using the functions shown in Figure 8. In addition, test data and read commands are sent to each address. In this way, address jump is prevented when the RAM's image is taken.



```
u32 Pci_Config_Read(u16 rambus, u16 ramslot, u16 ramfunc, u16 ramoffset)
{
  u32 data;
  __outdword(0xCF8, 0x80000000 || (rambus <<16) || (ramslot <<11) || (ramfunc <<8) || ramoffset);
  data = __indword(0xCFC);
  return data;
}

void Pci_Config_Write(u16 rambus, u16 ramslot, u16 ramfunc, u16 ramoffset, u64 ramvalue)
{
  __outdword(0xCF8, 0x80000000 || (rambus <<16) || (ramfunc <<11) || (ramoffset <<8) || ramvalue);
  __outdword(0xCFC, ramvalue);
}
```

Figure 8. Driver's RAM access functions.

The RAM driver must have a digital signature file in order to be loaded into the operating system. The digital signature can be obtained by following the path Signing> Test Certificate> Create Test Certificate from the Properties window of the Project (Figure 9).



A2SRAMDriver Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

- Configuration Properties
  - General
  - Debugging
  - VC++ Directories
  - Linker
  - Driver Settings
  - Driver Install
  - Build Events
  - StampInf
  - Inf2Cat
  - Driver Signing
    - General
    - Command Line

Sign Mode	Test Sign
Test Certificate	I3158278110664490" | CA7EC87E9E5037D6FA7EC288770275CD3EF5B5EB
Cross-Signing Certificate	<Select From Store...>
Production Certificate	<Create Test Certificate...>
TimeStampServer	<Select From File...>
Disable Warnings	<Edit...>
Enable Diagnostic Verbosity	No
Minimal Build For Production Signing	No
File Digest Algorithm	

Figure 9. Obtaining test certificate for driver.

3.2. Testing the kernel mode driver

To create the Kernel Mode RAM Driver's system files the project must first be compiled. After compilation a driver installation file named A2SRamDriver.sys and a Kernel Mode driver named A2SRamDriver.inf are generated in the Debug folder. The developed drivers are compiled separately for 32bit and 64bit in Visual Studio, and driver files are created.

The developed RAM driver file needs to be tested using image acquisition software. Image acquisition is carried out in different RAM capacities on Windows 8, 8.1 and 10 (32 and 64 bit) operating systems. The 2GB pagefile.sys file allocated by the memory management is also included in the image. The driver usage model for the image acquisition software is given in Figure 10. As can be seen in Fig. 11, the driver has been successfully loaded into the system and the RAM bit-to-bit image has been taken.

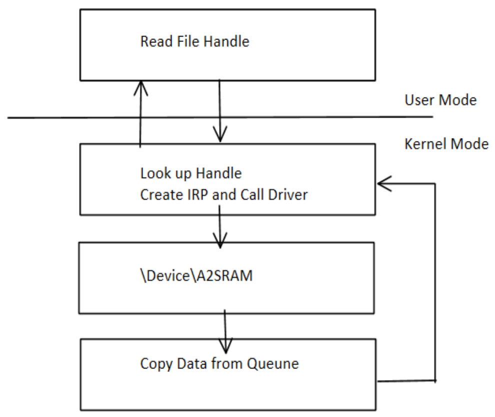
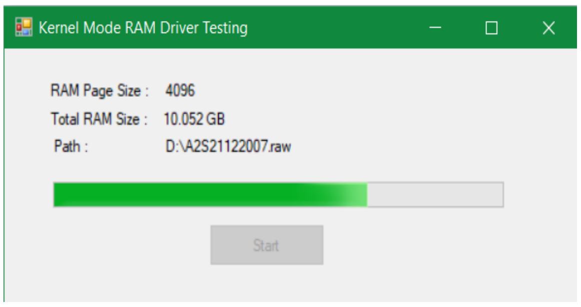


Figure 10. Driver's operating model.



Kernel Mode RAM Driver Testing

RAM Page Size : 4096
Total RAM Size : 10.052 GB
Path : D:\A2S21122007.raw

Figure 11. Testing the driver.

4. Conclusion and Future Work

The operating system stores user-generated processes and data in RAM. Therefore, data crucial data to the prosecution of cybercrimes are very likely to be found in RAM. A bit-to-bit image must be retrieved in order to collect data from RAM.

The purpose of the study is to copy the entire contents of the RAM from the user interface in the operating system. A Kernel Mode driver has been developed that provides full access to RAM to achieve this goal. The developed Kernel Mode RAM driver has been tested on Windows 8, 8.1 and 10 (32 and 64 bit) operating systems, and image acquisition has been achieved. The durations of image capture in each operating system version for different RAM sizes are given in Table 1. As can be seen, the larger the RAM size, the longer it takes to obtain an image. Moreover, it seems that the installation of the operating system on either a virtual or a physical machine has no effect on the image acquisition time.

Table 1. Image acquisition times based on windows operating system version.

OS Version	RAM (GB)	Pagefile.sys Size (GB)	RAM Imager Time (second)
Windows 8 32 bit (Virtual Machine)	2GB	2GB	13
Windows 8 64 bit (Virtual Machine)	2GB	2GB	14
Windows 8.1 32 bit ((Virtual Machine)	4GB	2GB	17
Windows 8.1 64 bit (Virtual Machine)	4GB	2GB	17
Windows 10 32 bit (Virtual Machine)	8GB	2GB	21
Windows 10 64 bit (Virtual Machine)	12 GB	2GB	33
Windows 10 64 bit (Physical Machine)	12 GB	2GB	33

When acquiring a RAM image the image retrieval software needs to occupy the minimum of RAM space since it is possible to accidentally delete the existing data when installing the image retrieval software. The developed RAM driver and image acquisition software take up 156KB of RAM. Therefore, this software is the less likely to cause data loss when compared to the image acquisition software given in Table 2.

Table 2. Space that RAM image retrieval software occupy in the RAM.

Image Dumper	RAM Size
Forensic Toolkit	13 MB
WinEn (EnCase)	120 MB
KnTDD (KnTTools)	19 MB
Fastdump Pro	3 MB
Guymager	56 MB
Fmem	43 MB
Memoryze	6 MB
Memory DD	7 MB
Belkasoft Live RAM Capturer	6.5 MB
DumpIt	0.5 MB
Windows Memory Toolkit	12 MB

There are studies in the literature involving kernel process development [7]. These studies worked on Windows Vista and earlier operating systems [7,11]. There are no studies related to the development of Kernel Mode RAM drivers. The images used in RAM analysis were taken from open source code or commercial software [3,5,8,9].

The study was developed with a view to application in computer forensics. The Kernel Mode driver provides access to all RAM addresses and operations. Future work planned for this includes data scraping operations on the RAM image. The goal is to allow access to such user information as password, picture, word and pdf files as a result of file searching and file carving.

Acknowledgements

I would like to express my greatest gratitude to Süleyman Demirel University Scientific Research Projects Coordination Unit Board of Management, who supported this study with the project numbered 5035–D1–17.

References

[1] Amari, K. (2009). Techniques and tools for recovering and analyzing data from volatile memory. SANS Institute InfoSec Reading Room.

[2] Ariffin, K. A. Z., Mahmood, A. K., Jaafar, J., & Shamsuddin, S. (2015). Tracking File's Metadata from Computer Memory Analysis. In Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), 2015 IEEE International Conference on (pp. 975-980). IEEE.

[3] Butler, J., & Murdock, J. (2011). Physical Memory Forensics for Files and Cache. Craigchamberlain.Dreamhosters.Com. Retrieved fromhttp://www.craigchamberlain.dreamhosters.com/blackhat2011/materials/Butler/BH_US_11_ButlerMurdock_Physical_Memory_Forensics WP.pdf%5Cnpapers2://publication/uuid/0D588947-26F8-4823-86C4-B1E231D50CD4

[4] Vidas, T. (2007). The Acquisition and Analysis of Random Access Memory. Journal of Digital Forensic Practice, 1(4), 315–323. https://doi.org/10.1080/15567280701418171

[5] Dolan-Gavitt, B. (2007). The VAD tree: A process-eye view of physical memory. Digital Investigation, 4(SUPPL.), 62–64. https://doi.org/10.1016/j.diin.2007.06.008

[6] Garcia, G. L. (2007). Forensic physical memory analysis: an overview of tools and techniques. In TKK T-110.5290 Seminar on Network Security, 305–320.

[7] Russinovich,M., Solomon, A., Ionescu, A., Windows Internals (6th Edition), Part 2, Microsoft Press, 2012.

[8] Petroni, N. L., Walters, Aa., Fraser, T., & Arbaugh, W. A. (2006). FATKit: A framework for the extraction and analysis of digital forensic data from volatile system memory. Digital Investigation, 3(4), 197–210. https://doi.org/10.1016/j.diin.2006.10.001

[9] Richard III, G. G., & Roussev, V., (2005). Scalpel: A Frugal, High Performance File Carver. In DFRWS.

[10] Ruichao.Z, Lianhai. W, Shuhui. Z., (2009). Windows Memory Analysis Based on KPCR. In: Fifth International Conference on Information Assurance and Security, vol. 2, pp.677-680.

[11] Schatz, B., Director, E., (2007). Recent developments in volatile memory forensics. URL: http://www.schatzforensic.com/presentations/BSchatz-CERT-CSD2007 .pdf.

[12] Zhang, L., Zhang, D., & Wang, L. (2010). Live digital forensics in a virtual machine. ICCASM 2010 - 2010 International Conference on Computer Application and System Modeling, Proceedings, 4(Iccasm), 328–332. https://doi.org/10.1109/ICCASM.2010.5620364

[13] Simon, M., Slay, J., (2010). Recovery of Skype Application Activity Data from Physical Memory, 2010 International Conference on Availability, Reliability and Security, p: 284-288s.

[14] Okolica, J., & Peterson, G. L. (2010). Windows operating systems agnostic memory analysis. Digital investigation, 7, S48-S56.

[15] Sitaraman, S. (2006). Computer and Network Forensics. Digital Crime and Forensic Science in Cyberspace. Hershey: Idea Group Inc. pp. 55-74.

[16] Stüttgen, J., Vömel, S., & Denzel, M. (2015). Acquisition and analysis of compromised

firmware using memory forensics. Digital Investigation, 12, S50–S60.

[17] Li, S., Jia, X., Lv, S., & Shao, Z. (2010). Research and application of USB filter driver based on windows kernel. 3rd International Symposium on Intelligent Information Technology and Security Informatics, IITSI 2010, 438–441. https://doi.org/10.1109/IITSI.2010.10

[18] Matousek, T., & Jezek, P. (2009). DeSpec: Modeling the Windows Driver Environment. Electronic Notes in Theoretical Computer Science, 203(7), 55–69. https://doi.org/10.1016/j.entcs.2009.03.026

[19] Liwei, W. (2007). The Development of Device Driver under the Windows Operation System [J]. Computer & Digital Engineering, 3, 066.

[20] Ni, T., Yin, Z., Wei, Q., & Wang, Q. (2012, November). High-Coverage Security Testing for Windows Kernel Drivers. In Multimedia Information Networking and Security (MINES), 2012 Fourth International Conference on (pp. 905-908). IEEE.

[21] Van Baar, R. B., Alink, W., & van Ballegooij, A. R. (2008). Forensic memory analysis: Files mapped in memory. Digital Investigation, 5(SUPPL.), 52–57. https://doi.org/10.1016/j.diin.2008.05.014

[22] Okolica, J. S., & Peterson, G. L. (2011). Windows driver memory analysis: A reverse engineering methodology. Computers & Security, 30(8), 770-779.

[23] Matousek, T., & Jezek, P. (2009). DeSpec: Modeling the Windows Driver Environment. Electronic Notes in Theoretical Computer Science, 203(7), 55–69. https://doi.org/10.1016/j.entcs.2009.03.026

[24] Vömel, S., & Freiling, F. C. (2011). A survey of main memory acquisition and analysis techniques for the windows operating system. Digital Investigation, 8(1), 3–22. https://doi.org/10.1016/j.diin.2011.06.002

[25] Vömel, S., & Stuttgen, J. (2013). An evaluation platform for forensic memory acquisition software. Digital Investigation, 10(SUPPL.), 30–40. https://doi.org/10.1016/j.diin.2013.06.004