Memory Acquisition by Using Network Card

Štefan Balogh

Štefan Balogh, Slovak University of Technology, Faculty of Electrical Engineering and Information Technology, Ilkovičova 3, Bratislava SK-812 19, Slovak Republic, stefan.balogh@stuba.sk

Received 4 March 2014;; Accepted 27 April 2014;
Publication 2 June 2014

Abstract

To detect present rootkit the rootkit and malware detectors need to have memory access. But, sophisticated rootkits are able to subvert the verification process of security scanner using virtual memory subversion techniques to hide their activity. We have proposed a new solution for direct memory access, based on a custom NDIS protocol driver that can send (on request of the local executable program) the contents of the computer memory over the network. Our method allows an unexpected type of the direct memory access, which is independent of the processor, and its control capabilities. This is a strong advantage in rootkit detection, because the rootkit cannot take any action to hide itself while the memory is scanned.

Keywords: Live Forensics; Memory Acquisition; DMA; Forensic analysis; network card; direct memory access; rootkit detection.

1 Introduction

Rootkits have become very sophisticated over the past few years. In 2006 the prototypes of rootkits that can subvert even the operating system by targeting hardware and firmware were presented in [1]. We have seen a surge in rootkit deployments in spyware, worms, and botnets. Generally, there are two types of rootkits: persistent rootkits and memory-based rootkits. Unlike persistent rootkits which can be loaded from disk into memory after reboot, in-memory rootkits make no effort to permanently store their code on disk or hook into

Journal of Cyber Security, Vol. 3 No. 1 , 65–76.
doi: 10.13052/jcsm2245-1439.314

the boot sequence. Their code exists only in volatile memory and they may be installed covertly via a software exploit. Thus kernel rootkits can control the execution path of kernel code, alter kernel data, and fake system call return values. Although once a computer system has been subverted by a rootkit it is extremely difficult to detect or eradicate the rootkit, there are still some different methodologies that detect the rootkit that have worked to varying degrees. To detect the kernel rootkits, we can use different techniques.

Scanning a signature of the rootkit during its in-memory execution, are worth mentioning because they have been applied with success to scanning system memory in addition to file system scanning. Ironically, most public kernel rootkits are susceptible to signature scans of kernel memory. It also can solve the problem with camouflaged binaries (by using a packing routine). But, these detection technique are useless against malware and rootkits for which a known signature does not exist. More-sophisticated rootkits are able to subvert the verification process by presenting an unmodified copy of the file for inspection, or by making modifications only in memory. So, memory scan based detection methods are useless against Virtual Memory Manager (VMM) hooking rootkits like Shadow Walker which are capable of controlling the memory reads of a scanner application [2]. Shadow Walker, can fake the contents of memory seen by other running applications. When the detector attempts to read any region of memory modified by the rootkit, it sees a 'normal', unaltered view of memory. To detect this kind of rootkits the memory scanner cannot depend on the OS API function or on processor control mechanisms.

Forensic analysis fights with similar limitations as rootkit detection: live detection can almost always be defeated by resident rootkits [3]. On the other hand offline analysis of the memory allows an investigator to see the state of the operating system without the operating system as a filter. Unlike trying to find a rootkit on a live system, the rootkit is unable to take any action to hide itself in the memory image. Therefore, investigators can see the data without the operating system, or the rootkit, interpreting the data for them [4].

There are also other methods today for rootkit detection. The earlier mentioned Signature based detection, or Heuristic / Behavioral detection (used by tool like VICE or Patchfinder [5]). Relatively new is cross view based detection which show a lot of promise. But, its success depends in large part upon implementation, specifically the method which is used to obtain the "low level" view of the system. Integrity based detection provides an alternative to both signatures and heuristics. However, the integrity checker is usually not capable of pinpointing the origin of the activity that has caused the changes.

Kernel rootkits can be especially difficult to detect and remove, because they operate at the same security level as the operating system itself and thus are able to intercept or subvert the most trusted operating system operations. To detect this generation of rootkits we need access to memory. The need for memory scanners are not confined to rootkits, but can also help to fight techniques that viruses started use since 2001, e.g. worm W32/CodeRed or W32/Slammer. These worms don't attack the files and computer disk. The malicious code can be executed before it is saved to disk. The worm is active only in memory and does not create any file. For this reason, neither files scanners nor on-access scanners (with reactive approach) are able to detect these attacks [6].

All existing rootkit detection techniques, it becomes apparent have strengths and weaknesses. The first step for rootkit and memory malware detection is scan the physical memory of the computer looking for rootkit behavior such as hooks in the SSDT, alterations to kernel functions (using kernel integrity checks), and modifications to key data structures like those performed by a DKOM attack. The tool for memory scan must remain as independent of the potentially subverted operating system as possible. To achieve this we:

- need access to memory (into different regions) without OS intervention
- must be able to make copy of the whole memory (memory dump – forensic analysis)
- and it's better to store the memory dump or part of memory copy in another computer)

In this article we have proposed a new method for scan the contents of the memory. The contents of memory are read by using DMA, so we avoid the possibility to change the contents by the running code. It could be very helpful for forensic purpose and also for rootkit detection.

The article is divided as follows: in the next section we present an overview of the methods for memory acquisition. Then we describe the existing attack that accesses the memory using a network card. In the third section we describe our new approach. Finally, we discuss the limitations and possible future improvements of the method.

2 Memory Access

Typically, there are two main approaches to memory access: software and hardware oriented, respectively. Software oriented tools use the fact that many operating systems allow reading the contents of memory: virtual devices

dev/mem and dev/kmem in UNIX, and DevicePhysicalMemory in Windows. However, even the act of running a memory dumping tool itself changes a portion of RAM. When the program is loaded into memory, it can (due to memory paging) move useful information to the page file [7]. But, the result, seen by the tools, can by modified by resident rootkits. Due to these shortcomings, software solutions for memory acquisition are not reliable.

The main idea of hardware oriented approaches is to bypass the operating system using a physical device. The device creates a memory access through a single channel, which is independent of the operating system. Very often firewire interface [8] or DMA (Direct Memory Access) feature is used for direct access. This allows us to obtain a memory image without launching another process or having to use potentially contested features of the local system. A concept of special-purpose PCI device can be used either for forensic purpose [9] or for rootkit detection [10], [11]. But, these devices either must, be already present in the system at a time, when we need to access the memory contents [9], or need to support stand-alone mode [10] (which may not be supported by many commercial network cards).

Several attacks to gain control of the DMA, and read the contents of RAM were presented. Almost exclusively these attacks are based on network cards. A network card uses DMA, and also allows a network access, so the content can be sent directly to the recipient. The first attack of this type was presented by a team from the French Office for Information Security (Agence Nationale de la Sécurité des Systèmes d'Information, ANSSI) in 2010 [12]. They abused the vulnerability of network cards from Broadcom, which had an incorrect implementation of ASF (Alert Standard Format) version 2.0, and exploited a bug in RISC RX (receiving parts of the network card).

Another way to obtain an image memory by the NIC was presented in 2011 by Guillaume Delugré [13]. It uses the fact that if we are able to run modified firmware on computers with network cards from Broadcom, we are able to control the packets, and thereby control what is read and written using DMA. The attack itself starts by changing the BD used by DMA write. Delugré found how DMA write can change the pointer in the memory, size of the data and flags. Necessary condition for this attack is to have a modified firmware for the adapter, which is not publicly accessible. In order to be able to send data remotely we need also, a special firmware with ASF support.

In 2011 Jiang Wang et.al [14] proposed firmware assisted method to reliably acquire the memory. They combined a commercial PCI network card (which is widely available) with the System Management Mode (SMM) to acquire the memory and CPU registers. They also change the SMM code,

and to solve the problem with compromising the network cards drivers by malware, they put the drivers into the SMM. But, in real we can write our own code and load it into the SMM only for some old machines, where the SMM is not locked by the BIOS. For new machines, the SMM is typically locked by the BIOS [14].

3 Memory Copy Using NDIS Driver

Our ultimate goal is to have a generic tool that can be used to read the memory contents for detection of rootkits, or for forensic purposes. None of the previous approaches is currently quite suitable due to its complexity, or a limited availability (FireWire). Therefore, we decided to create our own system. Our inspiration comes from Delugré attack, but we have chosen a different way to initialize DMA transfer, concretely the superstructure of network adapters NDIS (Network Driver Interface Specification). NDIS protocol driver can be used to service any network card.

The main goal of the memory access code is to create the NDIS driver packet with header and data section. The header contains the necessary parameters according to the used communication protocol (IP, TCP, UDP) and the data section contains the requested part of the memory. The packet is sent to the network driver and the network driver by using a standard communication

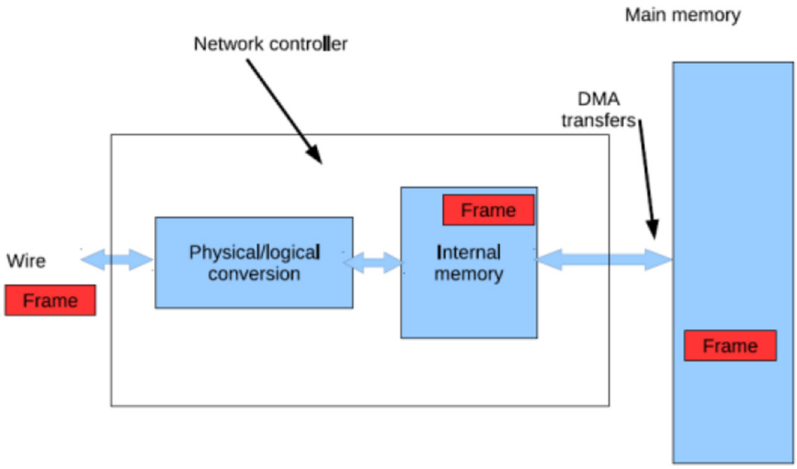

Figure 1 Ilustration of how DMA is used when network driver communicates with the network card (NIC) [13]

channel through DMA (see Figure 1) sends this packet to NIC. This DMA transfer avoids the possibility to change the contents of the memory by OS API functions or by processor control mechanisms when OS is compromised by malware.

We can use a remote control program or a local control program installed on the target computer to set up the necessary parameters for packet sending (IP address or MAC address where the memory content will be sent, and a required memory area, respectively). The remote control program can communicate with NDIS driver through the network. We have created a simple custom protocol for the communication between the control program and the driver.

3.1 NDIS Driver

NDIS driver is an interface with the main purpose of providing a standard API for the network interface card (NIC). It acts as an interface between the second and the third layer. The basic types are a protocol driver, an intermediate driver, and a miniport driver, respectively. Each has a different function and operates at a different level of the packet sending process. Network traffic, which is accepted by the NIC is controlled by the miniport driver. The protocol driver implements various protocols such as TCP/IP protocol. Finally, the intermediate driver is a hybrid between the previous two mentioned types. It is located between the Data link and Network layer and can control the operations accepted by NIC.

The intermediate and miniport driver has the possibility to control all incoming packets. So if a special packet is created and sent to the controlled computer, the NDIS driver will detect this packet, and setup the parameters, respectively. In this way we can remotely control the computer. For our practical implementation we have chosen the protocol driver, which is sufficient to provide a desired type of memory access. Also, remote control is not required in this test version. NDIS packets within the driver are declared as a structure of type NDIS_PACKET. We do not go into details, as the individual parts of that structure would not be treated directly.

The basic principle of operation of the system in the NDIS packet could be described as follows:

The NDIS_PACKET structure serves as a guiding structure which contains the basic information about the composition of the data. It is connected to the data in a buffer declared as NDIS_BUFFER (real data from higher layers to be sent), see Figure 2. Each NDIS_PACKET can be connected to several NDIS_BUFFER's.

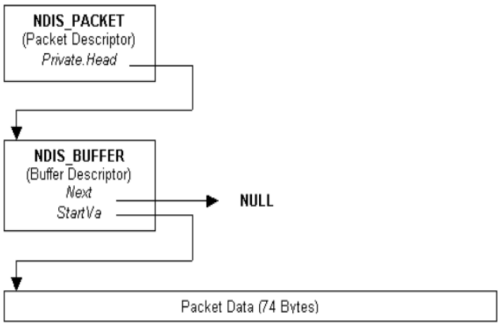

Figure 2 A simple version of NDIS buffer chaining [15]

3.2 Getting Memory Content

NDIS_PACKET structure contains information about connected buffers. Address of the sending packet in memory is located in NDIS_BUFFER structure. By using Ethernet II frame and IP protocol our packet has its own header and data section. Our goal is to replace the data section with a specified memory area, but preserve the Ethernet header (otherwise the packet will be lost in the network).

We divide a buffer to 2 smaller ones, and so we are able to work separately with the data, and with their memory space, respectively. The basic principle of the proposed approach is in Figure 3.

The attack proceeds as follows:

1. A new buffer pNdisBuffer1 is created. It has 14 bytes. In other words, we create a buffer in which the Ethernet header is stored.
2. When the data are separated from the header, we can modify them in such a way that they point instead of the Ethernet packet data to a designated address in the memory.
3. The buffer variable (pointer) that stores the address of data that contains the buffer is called MappedSystemVa. When we change its value to some arbitrary location in memory, the system will think that it contains the actually allocated data.
4. Setting the variable MappedSystemVa from buffer pNdisBuffer2 to the address of the beginning of memory space, or another requested memory location.
5. We are preparad for sending a packet with a selected chunk of memory. It remains only to combine the parts into one unit, which is done by concatenating pNdisPacket with pNdisBuffer1, and pNdisBuffer2.

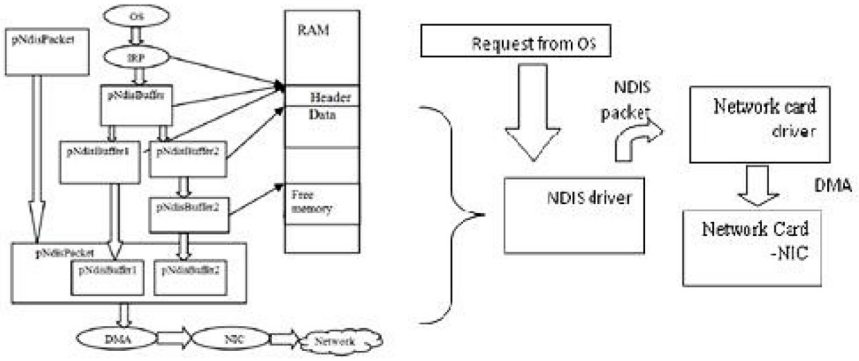

Figure 3 Basic principle of reading and send the memory contents

6. Finally, it remains only to send the packet. This will be served by the network driver and DMA engine.

But, in this way we can acquire only the access the memory contents or get the whole physical memory image of the target machine. To analyze or protect the critical OS components and structures we need to know their physical address. Also, to understand the physical memory, we must translate it to virtual memory used by the OS and find out the semantics of the memory. Only with this knowledge we can more effective protect or monitor the OS. However, it is difficult to obtain the semantics of a memory dump without knowing the values of the CPU registers at the time that the dump was retrieved. For that purpose, two CPU registers are critical: IDTR and CR3 [14]. IDTR points to the current interrupt descriptor table (IDT) and CR3 points to the base address of the current page table. CPU registers can be obtained in our solution using software based method. For this purpose we can utilize our NDIS driver. We can create special functions for getting information about CPU registers and also for getting the physical address of critical OS components and structures. This information's can by send as special packet to remote control program or a local control program. To implement extension like this make this solution more useful. We are actually working on this kind of extension.

3.3 Testing

To verify that we can really read the contents of memory from the desired memory address, we can use a simple technique in which known data is stored in memory. If you read them correctly back, it means that you really

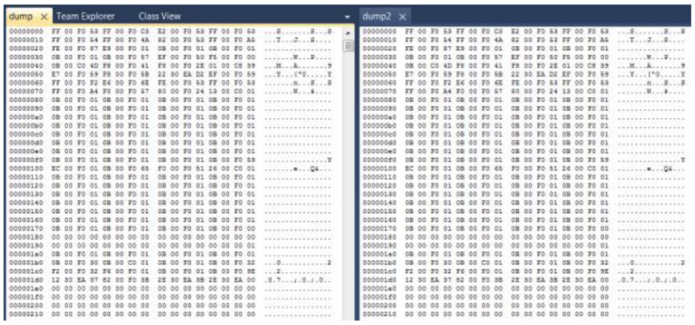

Figure 4 Memory dump: comparison of our solution and MDD tool

read your computer's memory. We try this test with data stored in known memory address with success. We have also tried to compare our dump with the results obtained by other existing tools. We made comparison, with dump of memory obtained by "MDD tool" [16]. MDD is capable of copying the complete contents of memory on computers with Microsoft Operating Systems. Both dumps were exactly the same (see Figure 4). These two kinds of tests indicate the possibility to get whole memory or only the specific part of the memory using our approach. However, this testing was performed on a clean PC without the rootkit. Testing with rootkits is more complex and must have correct methodology. We plan to test our method with the selected well-known rootkits of different types.

4 Conclusion

This paper deals with the study and implementation of direct memory access that satisfies the requirements for rootkit detection and for forensics analysis. In these cases we need to get the data from memory without a possible intervention of the operating system, or some rootkit installed on the computer. Therefore it is appropriate to implement a memory access in a way which is not controlled by the processor. Thus the rootkit cannot monitor and control the transfer, and the requested data. The obtained data are then stored outside the infected computer. The network cards seem to be the ideal solution in this case, as they have the hardware DMA into memory, and can immediately send the collected data over a network to a remote machine for the data analysis.

Our design follows this idea. We implemented a new method that allows us to dump the memory content using NDIS driver, and the network card (NIC). It would also be possible to replace the NDIS driver used in the attack from

protocol driver to intermediate or miniport driver. Then it will be possible to setup the NDIS driver not only by the control program installed on the local machine, but also through the network. The other advantage is that we can write, using NDIS driver, specific functions for getting the physical address of many critical OS components and structures. Using this information we can check the exact part of physical memory corresponding with protected or monitored system components (IAT / EAT / SSDT / IDT/ IRP tables, kernel functions (using integrity checks) or key data structures.

In this way, new possibilities for forensic purposes, as well as rootkit detection over the network can be enabled. Although the current implementation using protocol driver is still far from optimal, the imposed restrictions (such as the necessity to load the modified driver itself into a system) limits the potential misuse of this technique for illegal activities. Unfortunately, the installation of the modified driver can change the contents of the memory, and could not be used if a sophisticated rootkit (which knows about these tools) is already installed. To overcome this limitation, we can e.g. load the driver when the operation system starts. The best solution would be to implement this kind of driver already as a part of the operating system, as is suggested in [10]. It would still be necessary (and very difficult in practice) to ensure that the modified NDIS driver is not misused by unauthorized entities, and to implement proper security mechanism to protect the driver (we plan to test SMM mode described in [14] or some integrity check solution). Our DMA-enabling driver can be implemented in practice as a part of the data collection agent used in a more complex security solution.

Taking into account all advantages of the solution (access to memory using DMA, sending contents of the memory to other computers, control of the driver through network), it seems to be a promising approach for enabling memory access. The ability can be really helpful for forensics analysis, malware and rootkit detection, or as a generic basis for data collection agents but it must be more deeply tested. We plan to continue the development of the driver, as well as to test its forensic applications, in the future.

5 Acknowledgment

The publication was supported by Grant VEGA 1/0173/13.

References

[1] Chris Riesh. Inside Windows Rootkits, 2006, online: www.thehackademy .net/madchat/vxdevl/library/Inside Windows Rootkits.pdf

[2] Sparks, Sherriand Butler, Jamie. Raising The Bar For Windows Rootkit Detection, PhrackMagazine Volume 0x0b, Issue 0x3d, 2005.

[3] Michael Davis. Hacking Exposed Malware & Rootkits, McGraw-Hill, United States, Copyright, 2009 , ISBN 0071591192 / 97800715 91195.

[4] Jesse D. Kornblum , Exploiting the Rootkit Paradox with Windows Memory Analysis, International Journal of Digital Evidence Fall 2006, Volume 5, Issue 1, online: www.ijde.org .

[5] Rutkowska, Joanna. Detecting Windows Server Compromises with Patchfinder 2. January, 2004 online www.invisiblethings.org/papers/ rootkits_detection_with _ pat-chfinder2.pdf

[6] Szor, P. The Art of Computer Virus Research and Defense. Addison-Wesley Professional, 2005, ISBN 0321304543.

[7] Carvey, H.. 2009. Windows Forensic Analysis DVD Toolkit. 2. Edition. Syngress. June 11, 2009. ISBN-13: 978-1597494229, ASIN: 1597494224.

[8] Boileau, A.. 2006. "Hit By A Bus: Physical Access Attacks with Firewire" Security - As-sessment.com, Ruxcon, 2006. [cit. 2011–05–25]. Online: http://www.storm.net.nz/static/files/ab_firewire_ rux2k6-final.pdf.

[9] Carrier, B., Grand J.: A Hardware - Based Memory Acquisition Procedure for Digital In-vestigations. In Digital Investigation Journal. February 2004.

[10] N. L. Petroni, Jr., T. Fraser, J. Molina, and W. A. Arbaugh, "Copilot-a coprocessor-based kernel runtime integrity monitor," in SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium. Berkeley, CA, USA: USENIX Association, 2004, pp. 13–13.

[11] A. Baliga, V. Ganapathy, and L. Iftode, "Automatic inference and enforcement of kernel data structure invariants," in ACSAC '08: Proceedings of the 2008 Annual Computer Security Applications Conference. Washington, DC, USA: IEEE Computer Society, 2008, pp. 77–86.

[12] DUFLOT, Loïc, Yves-Alexis PEREZ, Guillaume VALADON a Olivier LEVILLAIN. Can you still trust your network card?. In: Agence nationale

de la sécurité des systèmes d'information, 2010 [cit. 2012–05–16]. Online: http://www.ssi.gouv.fr/IMG/pdf/csw-trustnetworkcard.pdf
[13] DELUGRÉ, Guillaume. Closer to metal: Reverse engineering the Broadcom NetExtreme's firmware. In: Sogeti ESEC Lab [online]. 2010 [cit. 2012–05–16]. Dostupné z: http://esec-lab.sogeti.com/dotclear/public/publications/10-hack.lu-nicreverse_slides.pdf
[14] J. Wang, F. Zhang, K. Sun, A.Stavrou. Firmware-assisted Memory Acquisition and Analysis tools for Digital Forensics, (SADFE), 2011 IEEE Sixth International Workshop, 2011
[15] PCA USA. NDIS developer's reference, 2012 [cit. 2012–05–16]. Online: http://ndis.com
[16] ManTech Memory DD (MDD) released under GPL by Mantech International http://sourceforge.net/projects/mdd/

Biography

Stefan Balogh has been an assistant professor at the Slovak University of Technology Faculty of Electrical Engineering and Information Technology, since 2007. He teaches classes in Information security, Communication protocols and Computer crime. He is completing his Ph.D. in information studies, where his research interest are in the areas of the forensic memory analysis, Cryptology and Behavior-Based Malware Detection.