International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-5 Issue-6, November 2015

Analysis and Detection of DMA Malware for Peripheral Devices

Abhinav Kharbanda, Meena Kumari

Abstract- Malware or malicious code aimed at exploiting information systems are continously evolving at a pace at which it becomes exacting to counter them. As the complexity of information systems and encryption techniques increases exponentially, the malwares developed to exploit the loopholes in them also become difficult to detect and comprehend. In this research paper, various innovative approaches to develop malware that can bypass existing counter measures to snoop and modify information present in system's primary memory or RAM, via Direct Memory Access (DMA), is analyzed. The exploits using DMA that easily dissemble from various end-user security mechanisms by executing their code on the processor and memory of the peripheral are described. The peripherals infected from DMA malware, if introduced in any one system, can spread across numerous inter-connected network systems in a data center, and hence have a devastating potential. The approach of exploiting systems using peripherals becomes pertinent because of the ability of a DMA malware to affect numerous users without being detected and the inadequacy of present counter-measures. The paper is concluded by describing major threats to information systems from malware installed on peripheral devices, executing stealthily and harnessing the advantage of a separate execution environment, perceptibly innocuous outlook, and DMA to host's primary memory.

Keywords: DMA malware, Direct Memory Access (DMA), Graphics processing unit (GPU), Malware, NIC, Peripherals, Rootkit.

I. INTRODUCTION

Computer peripherals - such as a keyboard, an optical mouse and a video card - are essentially defined as auxiliary devices that are connected to the host computer to provide additional functionality, but do not necessarily contribute to the primary function of computing. They can be external such as a keyboard, or internal such as a network interface card. Most peripherals have a dedicated hardware providing a seemless execution environment, secured and isolated from the host computer. The hardware consists of namely - a processor, ROM and a run-time execution memory in which commands stored in ROM are loaded and executed. Often this architecture is protected by firmware manufacturer's restrictions applied on the hardware. The presence of a separate execution environment makes the peripheral devices an efficient environment for the attacker to target and execute a malicious code. Since these devices are protected and isolated by the device manufacturer, most anti-virus software deployed on the host system remain oblivious to the malicious code present in EPROM of the peripheral.

Manuscript published on 30 November 2015.
*****Correspondence Author(s)
Abhinav Kharbanda, Student, Department of Computer Science, The North Cap University, Gurgaon, Haryana, India.
Dr. Meena Kumari, Professor, Department of Computer Science, The North Cap University, Gurgaon, Haryana, India.

Modern peripheral devices or I/O devices, for that matter, communicate with the host system using a method called Direct Memory Access (DMA). DMA allows the I/O device to communicate, monitor and send and receive data to or from host's primary memory without interrupting the CPU. This measure, speeds up memory operations without the need for any interference from the CPU. Most modern day peripheral devices have a DMA engine for employing direct memory access that is used to drive and control memory transfers by providing memory address range of source and destination, concurrency control and an interface between the bus of host CPU and the peripheral. Hence, the presence of an isolated execution environment, DMA access and an innocuous outlook of peripherals provide an opportunity to an attacker to maliciously monitor, modify and delete sensitive information stealthily, present on the host primary memory.

In section III, DMA malware is defined and in Section IV different approaches of DMA malware are analyzed. In section V, the counter measures used against DMA malware are illustrated and the paper is concluded in section VI. Prerequisites for this paper are described in the following section.

II. PREREQUISITE

To understand how DMA can be exploited for malicious purposes, It is essential to know the details of a 32 bit (x86) computer architecture. The details of the architecture are described to understand what restrictions a DMA malware needs to circumvent to maliciously snoop and modify information present on the primary memory of the host system. Further, the types of DMA engines, the required differences and reasons for choosing one over the other are described. To classify malware as a DMA malware, in the subsequent sections, the definitions of a malware and the characteristics for it to classify as a DMA malware are elaborated in this paper.

a. Typical x86 Architecture

In a typical x86 architecture [1], the chipset consists of a Central Processing Unit (CPU), an Input / Output Controller Hub (ICH) and a Memory Controller Hub (MCH). The MCH on the chipset is connected to a Random Access memory (primary memory or also known as RAM) for storing programs currently in execution. Further MCH is also connected to a display adapter or a video card for rendering display on a monitor. Most modern video cards have their own processor (Graphics Processing Unit or GPU) and their own run-time memory (VRAM).

The ICH interfaces with the peripherals such as the Network Interface Card (NIC), flash memory and other storage devices, using Peripheral Component Interconnect express (PCIe). The MCH controls the access to the memory (RAM). It interprets the memory access request, and can either block it or redirect to ICH, if the address requested belongs to the ICH or any other peripheral. A typical 32 bit (x86) architecture is depicted in Figure 1.

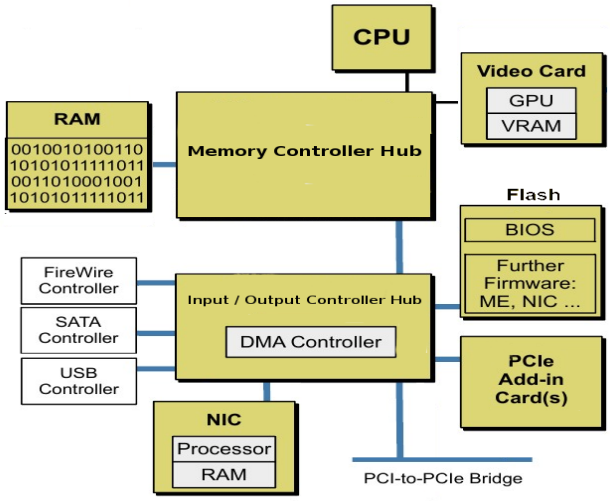

Figure 1: x86 architecture

b. Direct Memory Access

To communicate with the peripherals, a mechanism called Programmed I/O (PIO) is used. In PIO, the CPU uses input/output instructions to transfer data between the peripheral device and the primary memory of the host. One disadvantage to this scheme is slow transfer rates of I/O processes. CPU clock cycles are often much faster than the I/O cycles. Thereby, CPU cannot perform any other function while I/O transfers are taking place. Furthermore, In case of any data or resource hazards, CPU gets tied up indefinitely. To overcome this problem, a method known as Direct Memory Access is employed (DMA). DMA enables fast memory access without the involvement or interference from the host CPU, i.e. DMA allows peripherals to gain access to the address bus or whole host memory without interrupting CPU and therefore bypassing it. To employ DMA access, each architecture has a central DMA controller. The DMA controller takes control of the address and data bus, generates address, control and data signals to transfer the data between I/O devices and memory. This mechanism is called a Third Party DMA. Peripherals can have an inbuilt DMA engine to perform the function of DMA thereby eliminating the need of a central DMA controller. This mechanism is called First Party DMA. An illustration of first party and third party DMA engine is given in Figure 2 and Figure 3 respectively.

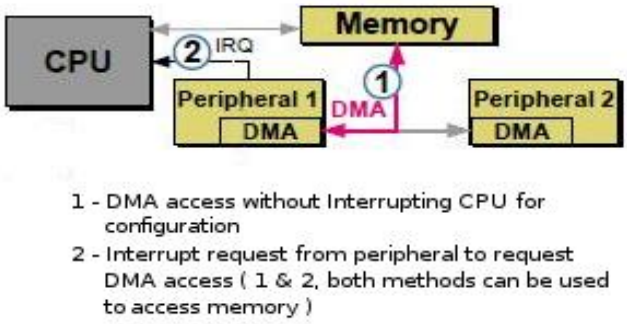

Figure 2: First party DMA engine

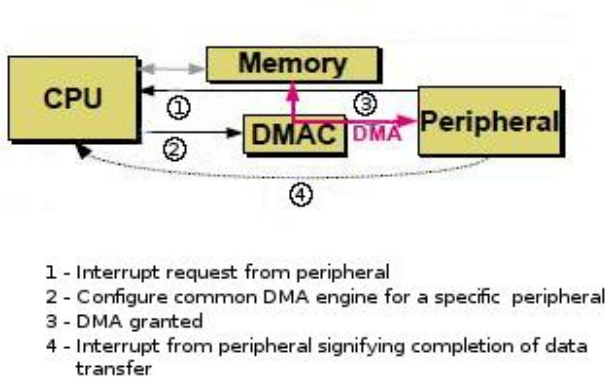

Figure 3: Third party DMA engine

c. Malware

Malware is known by various names such as a malevolent code, malicious code or a malicious software. Various definitions have been presented for describing malwares. According to C. Kruegel a Malicious code (or Malware) is defined as a software that fulfills the harmful intent of an attacker [2]. U. Bayer, C. Kruegel, and E. Kirda, together, define malware as a term used to denote all kinds of unwanted software (e.g, viruses, Trojan horses or worms) and such software pose a major security threat to computer users [3]. Christodorescu et al. state malware as a program whose objective is malevolent [4]. Fred Cohen defined a computer virus as: “a program that can 'infect' other programs by modifying them to include a possibly evolved copy of itself” [5]. G. McGraw and Greg Marrisett explained malicious code as any code added, deleted or changed in the system in order to deliberately cause harm or weaken the functionality expected from the system [6]. Ed Skoudis and Lenny Zeltser defined malware as a set of instructions that execute on your computer and make your system perform something that an attacker wants it to perform [7]. John Aycock in his paper “Computer Viruses and Malware” states malware as software whose effect is malicious or whose intent is malicious [8].

In this paper malware stands as a generic term that encompasses viruses, spywares, trojans and other intrusive code” given by A. Vasudevan and R. Yerraballi [9].

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

III. DMA MALWARE

DMA malware can be defined as a malware that attacks the host computer primarily by attacking the DMA engine and gaining access to snoop information and modify sensitive data present on host's primary memory (RAM). A DMA based malicious code portrays following characteristics:

1) It should implement malware functionality
2) It should not need any physical access to the host system to infiltrate stealthily
3) It should apply rootkit capability during run-time i.e. it should not be easily detectable by system software and counter-measures employed
4) It should survive reboot/standby/power off modes

A malware satisfying the above conditions as well as employing DMA for its attack on the Host system, can be classified as a DMA malware.

The CPU supports different privilege modes to provide protection from unauthorized execution of programs. The privilege modes are referred as rings and range from 0 to 4, ring 0 being the most privileged ring or mode, and ring 3 being the least. The operating system runs in ring 0 - with root privileges, whereas user applications and software run in ring 3 - with least privileges. Ring 1 is used for execution of device drivers and ring 2 for services [10].

DMA malware as stated above executes on ring-3 thereby it is also known as ring-3 rootkit.

IV. APPROACHES TO DMA MALWARE

In this section, the three approaches of creating a DMA malware are illustrated. The first approach solely uses the GPU for executing a Keylogger, whereas the second approach exploits the memory buffer of the NIC to store and further execute commands by mapping those to the host's primary memory. In the third approach a stealthy connection to a malicious remote host is established via NIC that allows the attacker to execute commands via SSH and snoop, modify and delete information present on host's primary memory.

a. GPU-Based Keylogger

In 2013, Evangelos Ladakis, Giorgos Vasiliadis, Michalis Polychronakis, Lazaros Koromilas, and Sotiris Ioannidis, developed a GPU-based stealthy Keylogger that monitors the keyboard buffer in every ~100ms interval for snooping key strokes. This is done by making use of GPU in NVIDIA CUDA devices. The malware has two basic components (i) A CPU-based component that executes once at the bootstrap phase, locates the address of the USB keyboard buffer in the primary memory of the host system (ii) A GPU-based component that monitors the keyboard buffer via DMA, and records all the keystrokes. The malware, first begins scanning the whole host memory for pointers to the USB device structures. These structures are memory-aligned to 0x400 boundaries of primary memory. It then searches for transfer_dma fields - the keyboard buffer is in the field transfer_buffer of the URB structure in linux. If the aforementioned conditions are true the malware checks whether the product field of URB contains the terms "keyboard" and "usb". In the final steps, the malware checks the transfer_buffer_length for approximately 8 bytes of data that contains valid keystoke values. In the second stage the malware proceeds to snoop and capture the keycodes. It does so by creating a virtual dummy page in the controller process - that manages the GPU - executing on host. After the scanning for keyboard buffer gets finished in the initial phase, the kernel module locates the page table of the controller process and remaps the virtual mapping of the dummy page to the physical page that contains keyboard buffer. By doing so, the GPU gets DMA access to keyboard buffer - which previously it did not have access to. The controller process then releases the page, thereby circumventing any anomaly detection tools - that check for any suspicious page table mappings. The GPU can now access the keyboard buffer, monitor and snoop keystrokes and store the key events in its device memory [11].

b. NIC-Based Malicious Code Execution

In 2010, Loïc Duflot, Benjamin Morin, and Yves-Alexis Perez, demonstrated a code execution strategy by overflowing the buffer of the username field, required to authenticate an RSP Session (Remote Management and Control security-extensions Protocol) with a Net Xtreme network card. The Net Xtreme network card has 3 main components in its architecture (i) PHY to send and receive signals from physical layer or wire (ii) DMA-Engine to communicate with host's primary memory (iii) Internal Memory (EPROM and SRAM) to load and store firmware and execution commands respectively. The NetXtreme network card supports 3 protocols namely - User Datagram Protocol (UDP), Ethernet Protocol, Internet Protocol (IP) and RSP. It supports two firmwares (i) ASF (Alert Standard Format protocol) (ii) TSO (TCP Segmentation Offloading). For demonstrating the exploit, researches took advantage of vulnerability present in the ASF firmware. The ASF firmware ver. 2.0 supports RSP by default. With the introduction of RSP (an iteration over RCMP) a remote user can share an RCMP message, authenticated by pre-shared HMAC-SHA1 keys over RSP and execute remote commands. With malicious RCMP commands, an attacker can stop the NIC from processing packets by crashing it, poison ARP/DNS of host system, replace firmware on NIC, read/write host's main memory.

The attack commences by sending the desired code to be executed (in future), in the form of normal IP packets. The first packet in these stream of packets contains a unique magic number to make it identifiable. The NIC stores these packets in its internal memory before transmitting it to the Host. The internal memory of the card is mapped to host's primary memory through a 32KB window. This means that if one is able to send an arbitrary code to NIC, one can execute it on host's memory. The second stage of the exploit commences by sending commands in the username field. The username field can only supports 255 chars, so instruction code is limited. The commands sent in the username field instruct the processor of NIC to search for the unique magic number in the IP packets stored inside internal memory of the card.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

By manipulating the stack using a debugger the researchers were able to over-write the return address in stack. When username field overflows with tries that consists of characters overflowing its limit of 255 characters, the stack returns to the memory address, over-written using debugger. This memory address is the starting address of the commands sent using username field in the starting of stage 2. When these commands are executed, they search for magic unique number in the device's internal memory and then having found the magic number, jumps to that address. The commands stored in internal memory, virtually mapped to the host's primary memory, can then be executed to read/write host's main memory, replace NIC's firmware, drop some packets or crash it completely. With this code execution using buffer overflow an attacker can maliciously manipulate and snoop sensitive information present on host's primary memory [12].

c. NIC-Based Snooping Attack with Remote Access

Arrigo Triulzi, under the project titled "Project Maux Mk.I" from 2006-2007 exploited the vulnerabilities in network interface card - Broadcom Tigon. In this iteration, Triulzi, using the SDK provided by Broadcom, developed a firmware for Broadcom Tigon and hooked IP checksum routines and a packet sniffer in the firmware. The firmware developed, had exiguous functionality of sniffing packets into a scratch RAM which had a capacity of storing approximately 5s-30s of sniffed packets. Also NIC in itself has slender primary memory which in turn made execution of instructions difficult. The lack of effective functionality led to the development of Project Maux Mk. II in the year 2008.

In 2008 under Project Maux Mk.II, Triulzi took the advantage of the processing power of NVIDIA GPU and implemented a remote shell functionality for remote code execution. For this he developed two firmwares (i) Firmware containing Maux code developed for NIC (ii) Firmware containing maux code developed for the GPU. The features in firmware developed for NIC were carried over from Project Maux Mk.I discussed above, with an additional support for grabbing the IP packet to check for the magic header and the functionality to forward the packet to GPU for execution via PCI-to-PCI transfer, if magic header was found.

The execution of the attack is two-fold. First, each IP packet is sniffed to search for the magic packet. The magic packet contains unique parameters for differentiating it from normal IP packets. These parameters include (i) IP ID of 0xbeef (ii) an IP address as 0x50b1463d (iii) IP timestamp option with flag value of 0x3 and (iv) a timestamp value of 0x06026860. If these parameters are found in an IP packet, the NIC transfers these packets to the GPU.

In the second stage, the firmware installed on GPU, contains quasi-SSH daemon (nicssh 1.0) persistently executing and waiting for the magic packet to arrive. The quasi-SSH needs no authentication or key exchange to authenticate an incoming session packet and provides a basic command shell with limited number of readline commands supporting tab completion and command history. It has capabilities to inspect the primary memory of the host and the GPU (VRAM). When the NIC forwards the magic IP packet to the GPU, the quasi-SSH interprets it and sends an appropriate ICMP reply. If the packet received is the first one to arrive, the daemon replies saying "Mauxed" to designate a mauxed system along with the details of the OS and its capabilities. In other case, if the packet is not the first one to arrive, the packet is interpreted as a part of an ongoing session.

In 2010, Project Maux evolved into "Project Maux Mk. III", also dubbed as "Jedi Packet Trick", providing implementation mechanisms for infecting multiple NICs present in a network by replicating its firmware and transmitting it over the network. This allowed the attacker to create a private communication channel between the NICs of various workstations in a network. Henceforth this communication channel allowed an attacker to access any workstation present on the network by just communicating with a single starting node and then forwarding the commands to the subsequent infected nodes over the private channel. With the help of an SSH daemon, the malicious execution of commands is able to circumvent any firewall or end-user countermeasures employed, with a "Mauxed" NIC as the OS remains completely oblivious to the communication channel established between NICs inside the network.This allows an attacker to snoop on the host's primary memory via DMA, and modify sensitive information, at the very least [13][14].

V. COUNTER MEASURES AGAINST DMA MALWARE

Intel issued a statement on September 2015, in response to a malware dubbed as "Jellyfish" developed for modern GPUs. The source code of the malware was released in May 2015 on Github [15]. The Jellyfish malware "leverages the LD_PRELOAD technique from the Jynx Linux rootkit and OpenCL" [16] as stated in official statement. The Jellyfish malware is currently designed to exploit computers with AMD and NVIDIA graphics card. However, Intel Security pointed out in its official statement that applications designed to access and execute on GPU need a parent process that should execute on the CPU. The activities performed by this CPU process can be monitored and flagged as malicious by security countermeasures deployed on the system. Intel also added that to deliver the payload containing the malware, physical memory of the Host must be mapped to the GPU, which requires an administrative/ root or ring-0 access thereby adding a footprint of malware on the host. Further, the requirement for administrative access put in place by the kernel can be circumvented by the malware, however by doing so the code executing on the GPU becomes orphaned. This, on a Windows system, leads to a process called TDR (Timeout Detection and Recovery) that resets the graphics card.

In today's world, where there is a constant war between hackers and security analysts and experts, GPU malware is widely prevalent, and if harbored to its full potential, it can lead to devastating results.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

However, as stated by Intel, even though GPU malware is difficult to detect and the detection surface reduces significantly because it executes by utilizing GPU's own internal primary memory (VRAM), the malware leaves trails of malicious activity during execution that can be detected and remediated by endpoint security products. The counter measures presently in use, to detect and mitigate the execution of a DMA malware are as follows:

1) **Measured Firmware** : A firmware attestation chip called Trusted Platform Module (TPM) developed by The Trusted Computing Group is installed on the chipset of a computer and stores integrity measurement parameters in the form of hash values computed from the binary code before code execution commences. This however implies that the checks are carried out during load time only, thereby making the counter measure vulnerable to circumvention. Also, this measure can not provide safeguard from transient attacks that take place in between two subsequent measurements.
2) **Signed Firmware** : This measure promotes the installation of firmware on the chip, signed only by the manufacturer itself. This measure however, does not include the possibility of run-time attacks.
3) **Latency Based Attestation** : The approach, presented by Yanlin Li, Jonathan M. McCune, and Adrian Perrig [17][18] requires the peripherals to not only compute a checksum value, but to also compute it in a specified time. Thereby an infected peripheral is identified if either the checksum value computed is wrong or if the checksum computation exceeded the specified time limit. This type of attestation requires the host to exactly know the hardware configuration of the peripheral device to be able to assign and attest an ideal checksum value to it. Also this doesn't work appropriately in the case of proxy attacks. In proxy attack an infected peripheral contacts a faster device to compute the correct checksum for it for time-sensitive checksum computations. This allows the malware infected peripheral to avoid detection by using a "proxy" peripheral [18].
4) **Monitoring Approach** : This approach proposes to monitor the firmware continuously during execution [12]. The run-time monitoring approach fall backs on performance with the need for 100% utilization of a CPU core.
5) **Bus Snooping Approach** : The researchers of this approach, Moon [6] and Lee [7], proposed a system that is able to snoop and monitor the memory for violations in kernel integrity. This approach though effective against transient attacks, requires a special hardware (for e.g an Intel i5 processor) that has computing power similar to that of the host system (also an Intel i5 processor). Through this approach, a system can detect the malware being transferred to the host memory via DMA thereby suggesting memory snooping approach to be applied only in case of write access from peripheral to host's primary memory. Hence, the proposed system is unable to prevent DMA malware exploiting read access where the attack comprises of reading and capturing sensitive information and cryptographic keys present in the host's primary memory.
6) **Input/Output Memory Management Unit (I/O MMU)** : With the use of I/O MMU, an I/O device is restricted to access the part of host's primary memory assigned to its Domain. The restrictions are applied using address translation tables. The I/O MMU configures DMA remapping (DMAR) engine for realizing these restrictions on a DMA access. I/O MMU can block a memory request if the device is not assigned to that block of memory addresses or domain. Though effective, I/O MMU is not reliable because of the necessity to configure it faultlessly. If not configured impeccably, I/O MMU can lead to conflicts in memory access policies. Furthermore these are not supported by every chipset and OS. I/O MMU significantly cause a performance overhead and can be switched off via BIOS in some Intel V-Pro processors.

As Yanlin Li, Jonathan M. Mccune, and Adrian Perrig rightly said, "In today's computer systems, all peripheral devices with firmware, such as network adapters, USB and disk controllers, and even the BIOS, are at risk from computer malware" [18]. The trend of malware on the firmware of peripherals will be a popular trend in the near future. This trend will become commonplace primarily because of the difficulty in detecting malicious code present in the peripheral's firmware. These computations become difficult because of two reasons (i) the limited memory and resources on peripherals make it difficult to employ complex security counter measures on peripherals (ii) hardware-based protection is unfeasible because of added exorbitant costs and complexity to devices [18].

VI. CONCLUSION

Attackers are continuously evolving malware to make them more undetectable and malicious. To counter these complex malwares, information security researchers are developing complex impregnable counter-measures. This paper explores and evaluates three techniques through which a malware could exploit the DMA access given to a host memory to snoop keystrokes, cryptographic keys, IP packets etc. These techniques primarily make use of NIC and GPU for computation needs and easily escape end-user counter-measures present today by executing on GPU's (or NIC's) independent processor and utilizing its run-time memory. This paper addresses the research challenge and predicament faced by the counter measures developed, and the reason for their pitfalls. This paper discusses a review of the techniques used for development and deployment of DMA malware and the perils they might cause in future. The paper concludes that the adversity caused by stealthy DMA malwares has the potential to spread uncontrollably as no suitable counter-measures are developed to preclude them and this potential combined with the innocuous outlook of peripherals can be catastrophic in future.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

REFERENCES

1. Stewin, Patrick, and Iurii Bystrov. "Understanding DMA Malware." Detection of Intrusions and Malware, and Vulnerability Assessment Lecture Notes in Computer Science (2013): 21-41. Web. <http://stewin.org/papers/dimvap15-stewin.pdf>.
2. Kruegel, Christopher. "Behavioral and Structural Properties of Malicious Code." Pub. In Advances in Information Security Malware Detection (2007). Web. <https://www.cs.ucsb.edu/~chris/research/doc/malware05_behavior.pdf>.
3. U. Bayer, C. Kruegel, and E. Kirda. “TTAnalyze: A Tool for Analyzing Malware.” In 15th Annual Conference of the European Institute for Computer Antivirus Research (EICAR), 2006. Web. <https://www.cs.ucsb.edu/~chris/research/doc/eicar06_ttanalyze.pdf>
4. M. Christodorescu, S. Jha, S.A. Seshia, D. Song, and R.E. Bryant, “Semantics-aware malware detection,” Proc. of the IEEE Symposium on Security and Privacy, 2005, pp. 32-46.
5. Cohen, Fred "Computer Viruses - Theory and of experiment" (1984). Web.<http://web.eecs.umich.edu/~aprakash/eecs588/handouts/cohen-viruses.html>
6. McGraw G, Greg Marrisett. “Attacking malicious code”. Submitted to IEEE Software and presented to Infosec Research Council. [Online](2000).Web.<http://www.cs.cornell.edu/home/jgm/cs711sp02/maliciouscode.pdf>
7. Skoudis, Ed, and Lenny Zeltser. ”Malware: Fighting Malicious Code.” Pub: Prentice Hall PTR, 2004. Web. <http://vxheaven.org/lib/pdf/Malware:%20Fighting%20Malicious%20Code.pdf>.
8. John Aycock, Computer Viruses and Malware (Advances in Information Security), Advances in Information Security, vol 22 - Springer, 2006
9. Vasudevan, A. and Yerraballi, R. “SPiKE: Engineering Malware Analysis Tools using Unobtrusive Binary-Instrumentation.” In Proc. Twenty-Ninth Australasian Computer Science Conference (ACSC 2006) pages 311–320.
10. Tereshkin, A., and R. Wojtczuk. "Introducing Ring -3 Rootkits." Proc. Of Black Hat conf., USA, 2009. Web. <http://www.blackhat.com/presentations/bh-usa-09/TERESHKIN/BHUSA09-Tereshkin-Ring3Rootkit-SLIDES.pdf>.
11. Ladakis, Evangelos, Lazaros Koromilas, Giorgos Vasiliadis, Michalis Polychronakis, and Sotiris Ioannidis. "You Can Type, but You Can't Hide: A Stealthy GPU-based Keylogger." Proc. of The 6th European Workshop on System Security. EuroSec, Prague, Czech Republic, Apr. 2013. Web. <http://www.cs.columbia.edu/~mikepo/papers/gpukeylogger.eurosec13.pdf>.
12. Duflot, Loïc. "Can You Still Trust Your Network Card?" Proc. of CanSecWest conf.., Mar. 2010. Web.<http://www.ssi.gouv.fr/IMG/pdf/csw-trustnetworkcard.pdf>.
13. Triulzi, Arrigo. "Project Maux Mk.II." Proc. of PACSEC conf., 2008. Web.<http://www.alchemistowl.org/arrigo/Papers/Arrigo-Triulzi-PACSEC08-Project-Maux-II.pdf>.
14. Triulzi, Arrigo. "Project Maux Mk.III." Proc. of Central Area Networking and Security (CANSEC) conf., 2010. Web. <http://www.alchemistowl.org/arrigo/Papers/Arrigo-Triulzi-CANSEC10-Project-Maux-III.pdf>.
15. Team Jellyfish, Project Jellyfish, (2015), GitHub repository, <https://github.com/x0r1/jellyfish>
16. Kovacs, Eduard. "PoC Linux Rootkit Uses GPU to Evade Detection." Security Week. Online Security Magazine, 08 May 2015. Web.<http://www.securityweek.com/poc-linux-rootkit-uses-gpu-evade-detection>.
17. Yanlin Li, Jonathan M. McCune, and Adrian Perrig. SBAP: Software-based Attestation for Peripherals. In Proceedings of the 3rd International Conference on Trust and Trustworthy Computing, TRUST’10, pages 16–29, Berlin, Heidelberg, 2010. Springer-Verlag.
18. Li, Yanlin, Jonathan M. Mccune, and Adrian Perrig. VIPER: Verifying the Integrity of PERipherals’ Firmware. In Proceedings of the ACM Conference on Computer and Communications Security (CCS), October 2011: n. pag. Web. <http://users.ece.cmu.edu/~jmmccune/papers/LiMcPe2011.pdf>.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)