© 2018 IJSRST | Volume 4 | Issue 5| Print ISSN: 2395-6011 | Online ISSN: 2395-602X
Themed Section: Scienceand Technology

Recognize and Monitor Kernel Virtualization Using Memory Heat Map

G. Soujanya Lakshmi[1], P.S. Naveen Kumar[2]

[1]PG Scholar, Department of MCA, St.Ann's College of Engineering&Technology, Chirala, Andhra Pradesh,India

[2]Assistant Professor, Department of MCA, St.Ann's College of Engineering& Technology, Chirala, Andhra Pradesh,India

ABSTRACT

Increasingly cyber attacks target the inner rings of a computer system, and they have seriously undermined the integrity of the entire computer systems. We focus on the threat posed by smart phone root kits. Root kits are malware that stealthily modify operating system code and data to achieve malicious goals, and have long been a problem for desktops. We propose, in this paper, an autonomic architecture called SHARK. Secure Hardware support Against Root Kit by employing hardware support to provide system-level security without trusting the software stack, including the OS kernel. Smart phones expose several unique interfaces, such as voice, GPS and battery that root kits can exploit in novel ways. The emergence of hardware virtualization technology has led to the development of OS independent malware such as the Virtual Machine based rootkits (VMBRs). We draw attention to a different but related threat that exists on many commodity systems in operation today: The System Management Mode based root kit (SMBR). System Management Mode (SMM) is a relatively obscure mode on Intel processors used for low-level hardware control the predictable nature of real-time embedded applications. We introduce Memory Heat Map (MHM) to characterize the memory behavior of the operating system. Our machine learning algorithms automatically summarize the information contained in the MHMs and then detect deviations from the normal memory behavior patterns. Normally kernel can be protected by using three different strategies which includes monitoring the invoked process snooping the incoming packets at network level and establishing trust of a process by using TCB(Trusted computing Base updated by the admin) different methods in different layer for example In network layer by snooping incoming packets.

Keywords: Kernel, OS, Process monitoring, Malware analysis, Virtual Machine Monitor, memory heat map, real-time systems, Malware, Virtualization, Operating System Security.

I. INTRODUCTION

Over the last tend the decreasing cost of advanced computing and communication hardware has access mobile phones to evolve into general purpose computing platforms. Over 115 million such smart phones uses worldwide in 2007 [1]. These phones are equipped with a rich set of hardware interfaces and application programs that let users interact better with the cyber and the physical worlds [2]. We show that smart phones are just as vulnerable as desktop operating systems to kernel-level root kits. Root kits are malware that achieve their malicious goals by infecting the operating system [3]. We propose a process context-aware architecture called SHARK is identify process contexts that are utilizing hardware resources without the OS' decree. By making use of such architecture system administrators can directly examine the feedback provided by the underlying hardware and compare it against the OS retrieved data

to know the state of the un-trusted OS [4]. In this work we focus on techniques to reveal malware applications malware virtual machines and hypervisors running in stealth. To the best of our knowledge this is the first effort using a synergistic micro architecture and OS technique to address the root kit exploits [5]. Location tracking with GPS presents a root kit that compromises privacy of a victim's location. When an attacker sends a command to a root kit infected phone the root kit queries the GPS device and sends the victim's coordinates to the attacker[6]. Denial of service via battery exhaustion smart phones is battery operated and is hence resource constrained. We demonstrate a root kit that stealthily exhausts a smart phone's battery. This attack renders the phone unusable when its user needs [7]. Unfortunately, these techniques are useless against Virtual Machine Based Root kits (VMBRs) which have the ability to exist independently of any OS. Such root kits are able to exert an alarming degree of control without modifying a single byte in the Operating System [8]. This process is invisible to the OS. Once installed the VMM is capable of transparently intercepting and modifying states and events occurring in the virtualized OS. The VMBR has virtualized memory its code footprint will also be invisible. These things make a VMBR extremely difficult to detect [9].

Figure 1. Architecture Overview of FB-DIMM

II. RELATED WORK

Root kit detection tools on desktop computer systems are largely centered on known techniques used by root kits to hide their presence [10]. Early root kits operated by replacing system binaries and shared libraries on disk with Trojan versions which would hide malicious objects owned by the attacker [11]. Memory subversion was first implemented the Shadow Walker root kit demonstrated that it was possible to control the view of memory regions seen by the operating System and other processes by hooking the paging mechanism and exploiting the Intel split TLB architecture [12]. Using these techniques it was capable of hiding both its own code and changes to other Operating System components. This enabled it to fool both signature and heuristic based scans [13]. Virtual-machine based root kits have many characteristics in common with the System Management Mode based root kit presented it hides its code footprint using memory virtualization supports nested virtual machine monitors, and implements countermeasures against timing based detections [14]. Our solution is used to the emerging DDR4 by integrating our new components with the DDR4 switch fabric a topic of future research [15]. For fully buffered DRAM it is crucial that we have to make sure that there is no performance penalty with the inspection of the memory traffic because such overhead will likely violate DRAM time constraint and render the solution useless [16].

III. SHARK ARCHITECTURE

After exploring possible architectural method it is evident that all the shortcomings were due to the tightly coupled dependency of these mechanisms with the OS itself which could have already been compromised [17]. As such detecting root kits with OS direct intervention will always fail to address this issue once and for all users to design processor architecture to be process context-aware [18]. We propose a novel processor architecture called SHARK, which stands for Secure Hardware support Against Root Kit. In a SHARK processor the master control of processes is delegated to the hardware for enforcing the security of process contexts [19]. Hardware Assisted PID Generation Process Page Table

Encryption and Decryption, and Process Authentication into one processor These mechanisms are implemented within the SHARK security manager, a hardware-based micro architectural extension while working seamlessly with the OS. The following sections detail each component in SHARK [20].

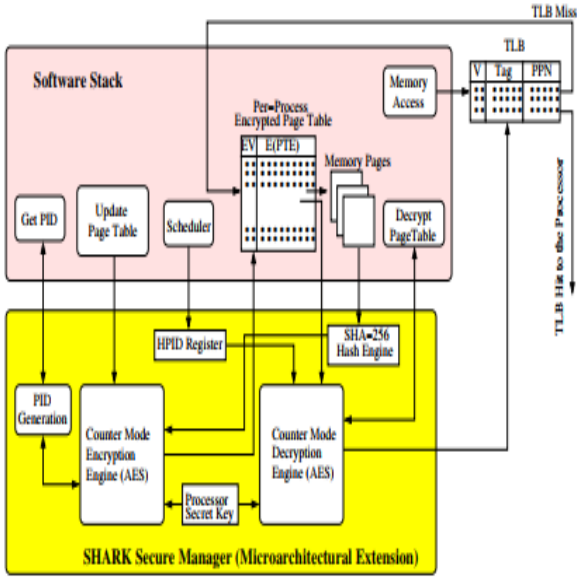

Figure 2. Architectural Support for SHARK Processor

IV. MEMORY HEAT MAP

A memory region is divided into cells, each with a size each cell counts the number of accesses to a region of size δ for a specified time interval. One can even consider it to be the temperature of each cell. The temperature of each cell on its own may not reveal useful information; due to variations caused by numerous factors the state of the entire map may reveal important system activities [21]. An MHM represents a composition of memory accesses from a variety of system activities due to applications and OS. MHM can be represented by a weighted combination of the primary activities; where the weights represent their contributions to the MHM [22].

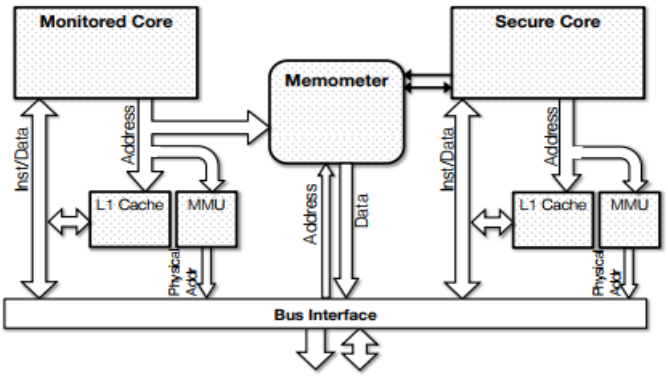

Figure 3. The secure architecture for memory using memometer

A. HARDWARE ASSISTED INTEGRITY MONITOR

Hyper Check is composed of three key components: the physical memory acquiring module the analysis module and the CPU register checking module. The memory acquiring module reads the contents of the physical memory of the machine and sends them to the analysis module [23]. It checks the memory contents and verifies if anything is altered the CPU register checking module reads the registers and validates their integrity [24]. We use hardware a PCI Ethernet card as a memory acquiring module and SMM to read the CPU registers. Usually, Ethernet cards are PCI devices with bus master mode enabled and are able to read the physical memory through DMA which does not need help from CPU [25]. SMM is an independent operating mode and could be made inaccessible from protected mode hence hypervisor and privileged domains cannot run [26]. PCI devices cannot read the CPU registers, thereby failing to detect this kind of attacks. By using SMM, Hyper Check can examine the registers and report the suspicious modifications [27].

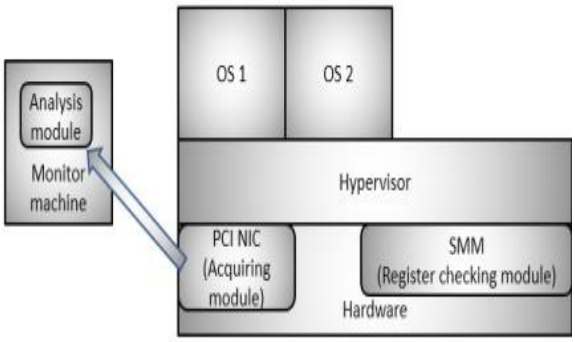

Figure 4. Hyper check

B. SMM Memory Space

The System Management Memory Space (SMRAM) is used to hold the processor state information saved upon an entry to SMM the SMI handler and its associated data [28]. The Intel chipset documentation defines three locations for SMRAM: Compatible, High Memory Segment (HSEG), and Top of Memory Segment (TSEG). Structurally the SMRAM space consists of a state save area and the System Management Interrupt (SMI) handler. The remaining space is available for use by the handler for data and stack storage. An internal processor register called SMBASE holds the physical address pointer to the start of the SMRAM space [29]. The SMBASE value is also stored in the state save area. Furthermore, the state save area is located at an offset from the beginning of SMRAM in physical memory.

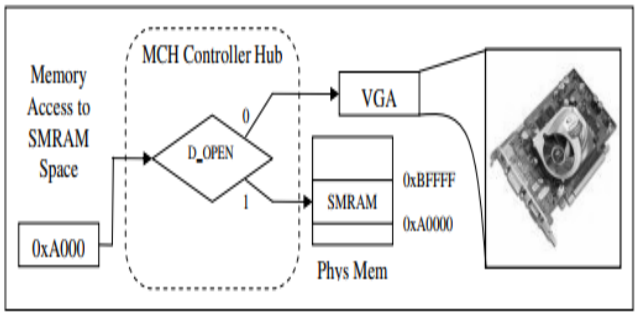

Figure 5. SMRAM memory accesses

1. On a host machine, an attacker makes SMRAM visible from security mode for reading and writing by setting the D_OPEN bit.

2. Once D_OPEN is set, the attacker copies the root kit SMM handler code to the handler security of SMRAM as defined by the Intel documentation.

3. Finally, the attacker clears the D_OPEN bit and sets the D_LCK bit. This has the effect of making SMRAM invisible to everything other than the subverted SMI handler and of locking the SMRAMC register so that it can no longer be modified.

C. ROOTKITS ON SMART PHONES

We present three proof-of-concept root kits that we developed to illustrate the threat that they pose to smart phones. We chose this platform because (a) Linux source code is freely available, thereby allowing us to study and modify its data structures at will; and (b) the Neo Free runner allows for easy experimentation it allows end-users to re-flash the phone with newer versions of the operating system [30]. We expect that these LKMs will be delivered via other mechanisms after an attacker has compromised a network-facing application or via a drive by-download attack presents the lines of code needed to implement each attack, and the size of the corresponding kernel module. it stealthily dials the attacker's phone is listen into or remotely record ongoing conversations. Alternatively, the root kit could trigger when the victim dials a number [31].

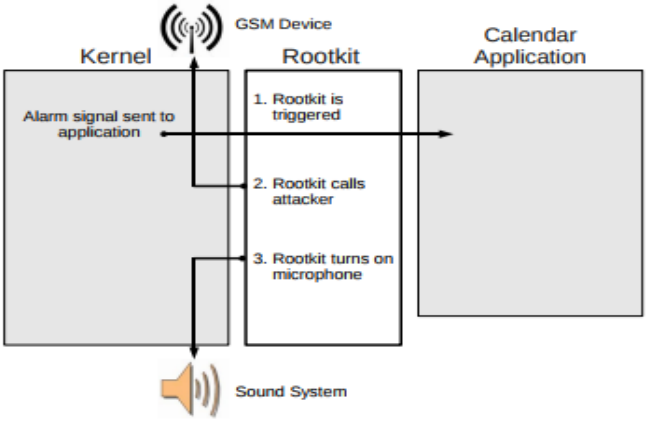

Figure 6. The GSM root kit intercepts

V. EXPERIMENTAL ANALYSIS

We conducted two sets of experiments to evaluate the proposed SHARK Security Manager. First, we evaluated the practicality and strength of the proposed scheme against malware running in stealth using real kernel root kits available on Linux. We performed

performance experiments to quantify the overheads incurred by the SHARK architecture. We obtained cycle information from VirtutechSimics with its cache model enabled. Staller will stall the cycle accounting mechanism whenever a cache miss occurs. The moment when the root kit is being loaded is distinguishable as expected after the launch the traffic does not show abnormalities in terms of the volume. This is because the root kit still calls the original read handler which resides in the region being monitored.

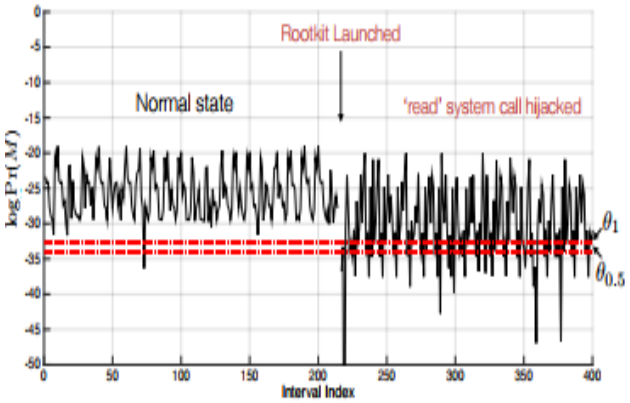

Figure 8. The root kit hijacks read system calls

VI. CONCLUSION

SHARK is process context aware it employs secure hardware support to provide system-level security without trusting the software stack, including the OS kernel. The proposed mechanisms including hardware PID, page table encryption, and process authentication, tightly couple the dependency between the OS and hardware architecture, making the entire system more security-aware. Under SHARK, the concealed malware at user, kernel and VMM levels of the software stack will be revealed automatically by the synergistic cooperation between SHARK and the software stack. Root kits evade detection by compromising the operating system, thereby allowing them to defeat user-space detection tools and operate stealthily for extended periods of time. We suggest that the emergence of such malware necessitates a shift in perspective from detection to prevention and that a closer relationship between security researchers and hardware developers should be fostered. We also plan to extend the architecture to support more than two cores and evaluate the required hardware changes, and to explore Deep Learning-based technique to deal with more complex embedded systems. Virtual machine security is very important in cloud computing, we have discussed various virtual machine architectures & its techniques to prevent malicious attacks in kernel.

VII. REFERENCES

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

[2]. Google fixes android root-access flaw news/security/0,39044215,62048148,00.htm.

[3]. Mcafee mobile security report 2008. research/mobile_security_report_2008.html.

[4]. National institute of science and technology fips pub 180-2: Sha256 hashing algorithm.

[5]. Rootkits,The Growing Threat, McAfee. rootkits1 en.pdf.

[6]. Vinod Ganapathy Arati Baliga and Liviu Iftode. Automatic inference and enforcement of kernel data structure invariants. In ACSAC '08: Proceedings of the Annual Computer Security and Applications Conference, 2008.

[7]. Arati Baliga, Liviu Iftode, and Xiaoxin Chen. Automated containment of rootkits attacks. Computers & Security, 27(7-8):323 - 334, 2008.

[8]. J. Rutkowska. Subverting Vista Kernel for Fun and Profit. Presented at Black Hat USA, Aug. 2006.

[9]. Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2. May 2007.

[10]. Arati Baliga, Liviu Iftode, and Xiaoxin Chen. Automated containment of rootkits attacks. Computers & Security, 27(7-8):323 - 334, 2008.

[11]. Arati Baliga, Pandurang Kamat, and Liviu Iftode. Lurking in the shadows: Identifying systemic threats

to kernel data. In SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy, 2007.

[12]. Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1. May 2007.

[13]. Intel Corporation. Intel 82801DB I/O Controller Hub 4 (ICH4). May 2002.

[14]. Intel Corporation. Intel 845GE/845PE Chipset Datasheet. Oct. 2002.

[15]. N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. The m5 simulator: Modeling networked systems. IEEE Micro, 26(4):52-60, July 2006.

[16]. Buffer. Hijacking linux page fault handler. Phrack Magazine, 0x0B, 0x3D, Phile #0x07 of 0x0f, 2003.

[17]. Intel. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1, 2007.

[18]. T. Kgil, L. Falk, and T. Mudge. Chiplock: support for secure microarchitectures. SIGARCH Computer Archititecture News, 33(1):134-143, 2005.

[19]. S. T. King, P. M. Chen, Y.-M. Wang, C. Verbowski, H. J. Wang, and J. R. Lorch. SubVirt: Implementing malware with virtual machines. In Proceedings of the 2006 IEEE Symposium on Security and Privacy, 2

[20]. P. Magnusson, M. Christensson J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A Full System Simulation Platform. IEEE Computer, Feb. 2002.

[21]. S. Chen, B. Falsafi, P. B. Gibbons, M. Kozuch, T. C. Mowry, R. Teodorescu, A. Ailamaki, L. Fix, G. R. Ganger, B. Lin, and S. W. Schlosser. Log-based architectures for general-purpose monitoring of deployed code. In Workshop on architectural and system support for improving software dependability, 2006.

[22]. J. Criswell, N. Dautenhahn, and V. Adve. Kcofi: Complete control-flow integrity for commodity operating system kernels. In IEEE Symposium on Security and Privacy, 2014.

[23]. H. Etoh. GCC Extension for Protecting Applications From Stacksmashing Attacks. Accessed May 2011.

[24]. Vendicator. Stack Shield: A "Stack Smashing" Technique Protection Tool for Linux. Accessed May 2011.

[25]. Bypassing Non-executable-stack during Exploitation using Return-tolibc. Phrack Magazine.

[26]. E. Buchanan, R. Roemer, H. Shacham, and S. Savage. When Good Instructions Go Bad: Generalizing Return-Oriented Programming to RISC. In Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08), pages 27-38. ACM Press, Oct. 2008.

[27]. Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: A virtual machine-based platform for trusted computing. In SOSP03: ACM Symposium on Operating System Principles, October 2003.

[28]. Tal Garfinkel and Mendel Rosenblum. A virtual machine introspection based architecture for intrusion detection. In Proc. Network and Distributed Systems Security Symposium, 2003.

[29]. M. Hypponen. The state of cell phone malware in 2007.

[30]. Nick L. Petroni Jr., Timothy Fraser, Jesus Molina, and William A. Arbaugh. Copilot - a coprocessor-based kernel runtime integrity monitor. In Security '04: Proceedings of the USENIX Security Symposium, 2004.

[31]. Gene H. Kim and Eugene H. Spafford. The design and implementation of tripwire: a file system integrity checker. In CCS '94: Proceedings of the 2nd ACM Conference on Computer and communications security,1994.

ABOUT AUTHORS:

G.SOUJANYA LAKSHMI is currently pursuing her MCA in MCA Department, St.Ann's College Of Engineering and Technology, Chirala, A.P. She received her Bachelor of science from ANU.

P.S.NAVEEN KUMAR received his M.Tech. (CSE) from jntu Kakinada. Presently he is working as an Assistant Professor in MCA Department, St.Ann's College Of Engineering &Technology , Chirala. His research includes networking and data mining