Firmware-assisted Memory Acquisition and Analysis Tools for Digital Forensics

Jiang Wang, Fengwei Zhang, Kun Sun, and Angelos Stavrou
Center for Secure Information Systems
George Mason University
Fairfax, VA 22030
Email: {jwanga, fzhang4, ksun3, astavrou}@gmu.edu

***Abstract*—Being able to inspect and analyze the operational state of commodity machines is crucial for modern digital forensics. Indeed, volatile system state including memory data and CPU registers contain information that cannot be directly inferred or reconstructed by acquiring the contents of the non-volatile storage. Unfortunately, it still remains an open problem how to reliably and consistently retrieve the volatile machine state without disrupting its operation. In this paper, we propose to leverage commercial PCI network cards and the current x86 implementation of System Management Mode to reliably replicate the physical memory and critical CPU registers from commodity hardware. Furthermore, we demonstrate how remote state replication can be used for semantic reconstruction, where the analysis of memory structures enables us to interactively perform forensic analysis of the machine's memory content.**

***Index Terms*—Live Forensics, Memory Acquisition, SMM-mode, PCI.**

I. Introduction

Analyzing volatile state information from a live, working system without disrupting its operation is critical for forensics because it offers a context for the static, non-volatile data. Indeed, by extracting the dynamic context, an analyst is empowered with a wide-range of information: from process description including currently running and stopped programs and their physical memory, CPU and Disk cache, network connections and open files, to operating system state such as system load, open sockets, and inter-process communication. This global view of the system state is crucial especially for stealthy malware that maintains a fully dynamic profile by operating entirely in memory. Instances of Code Red [1] and SQL Slammer [2] are two instances of such malcode. Therefore, being able to unobtrusively acquire the complete contents of the volatile memory of a computer is a necessary source of evidence for digital forensic analysis [3], [4], [5]. Currently, the majority forensic analysis tools employ the operating system, employ dedicated hardware to retrieve the running state of a system, or require the analysis of memory traces after shutdown [6].

Nevertheless, it is not trivial to reliably acquire the memory content of a running system without disrupting its operation. Many operating systems (OSes) provide software interfaces for the user or kernel level programs to read specific areas of memory. However, if the OS itself has already been compromised or the malware operates on the hypervisor level, the OS cannot be trusted to provide the correct RAM contents. Rootkits are notorious for hiding process information from user-land and sometimes kernel-level programs that read standard memory structures. Due to these shortcomings, software solutions for memory acquisition are not reliable. To address this limitation, researchers [4], [7] have turned their attention to hardware-based methods. For example, a customized PCI card can be used to read the physical memory via Direct Memory Access (DMA).

Unfortunately, using specialized PCI cards to access the volatile state has still two unresolved challenges. First, PCI cards can be blocked from access to all the physical RAM memory. Both Intel and AMD provide technologies, (VT-d [8] and IOMMU [9]) to restrict the memory access for peripheral devices as a security precaution. Although the purpose is to protect the privileged software from being compromised by malicious hardware, these technologies also thwart the PCI dependent tools to monitor the physical memory. Second, it is difficult to obtain the semantics of a memory dump without knowing the values of the CPU registers at the time that the dumb was retrieved. For instance, the interrupt descriptor table register (IDTR) points to the current interrupt descriptor table (IDT). Without knowing the value of the IDTR register, an analyst has to apply heuristic or pattern matching methods to search for and identify the value of IDTR. Of course, none of these methods are considered reliable and thus cannot be used to substantiate a robust forensic analysis. The same holds for a range of other CPU-derived information including the control register 3 (CR3), which points to the base address of the current page table.

To address some of these challenges, we introduce a firmware assisted method to reliably acquire the memory and CPU registers. In addition, we propose a memory analysis framework to assist the forensic investigator with the tedious task of analyzing the machine state. To that end, we design the capability to both interactively and automatically examine the contents of memory, CPU registers, and peripheral state remotely. We combine a commercial PCI network card (which is widely available) with the System Management Mode (SMM) to acquire the memory and CPU registers. SMM is a special CPU mode besides the protected mode and real-address mode [10]. The code of SMM, that resides in BIOS, can check the CPU registers and leverage the commercial PCI

network cards to read the memory. Therefore, we can read and transmit CPU registers, physical memory, and peripheral device state.

The memory and CPU information can be either communicated to a remote server to be examined off-line, or analyzed online (live forensics). For the live analysis mode, we can guarantee consistency because during SMM the OS enters and remains in suspended state. While in suspended state the system view remains untouched when being examined. To investigate the live memory contents, we propose to implement a GDB-like server and a GDB stub in SMM, so that it can be connected with a GDB debugger on another machine via serial console. The benefit of this SMM based online investigation is that it can examine the physical memory of user level programs and kernel code. Since the debugger running in SMM is isolated from the OS, it is much more reliable than other OS-based live forensic methods. Even if the whole OS, including the kernel, is compromised, the SMM is still protected and can reliably perform the investigation tasks.

II. SMM Background

The System Management Mode (SMM) is a separate CPU mode from the traditional protected and real-address mode. It provides a transparent mechanism for implementing platform specific system-control functions, such as power management and system security. SMM is primarily designed for firmware or basic input-output system (BIOS). System Management Mode (SMM) was first introduced in the Intel386 SL and Intel486 SL processors. It became a standard IA-32 feature in the Pentium processor [10]. SMM is a separate x86 processor mode from protected mode or real-address mode.

The motherboard controller is programmed to recognize many types of events and timeouts to trigger SMM. When such an event occurs, the chipset asserts the SMM interrupt pin (SMI#). At the next instruction boundary, the microprocessor saves its entire state and enters SMM. After the microprocessor's state has been stored to memory, the special SMM handler begins to execute. In SMM, the processor switches to a separate address space, named as system management RAM (SMRAM). All microprocessor context of the currently running code is saved in SMRAM. The SMRAM can be made inaccessible from other CPU operating modes; therefore, it can act as a trusted storage, sealed from being accessed from any device or even the CPU (while not in SMM mode). The RSM instruction is called to exit SMM. RSM reads the microprocessor state data from the SMRAM and restores the entire microprocessor state. From now on, the previous program resumes its execution from where it was interrupted.

Our memory acquisition and analysis system is implemented in the trusted SMM code (called SMRAM). This secure memory region is an aspect of SMM that it crucial for the secure and reliable implementation of any inspection system. Indeed, the SMM code remains locked from all the non-SMM CPU modes and thus, it can be safely used as a means of examining the system without any risks of being modified by a malicious user or program.

III. Architecture

The overall architecture of our system is depicted in Figure 1. The entire system is composed of two computers: one that is used for analysis and another that is the machine subjected to inspection. On the right is the target machine that is being investigated, which includes a dedicated network card and a serial port. On the left of the figure is the remote machine that is used by the forensics investigator. These two machines are connected by a network cable and a serial cable. The network connection is used for off-line investigation, the serial connection for on-line investigation.

A. Off-line Investigation

The off-line memory investigation collects the content of the physical memory and transfers it to the remote server. Then, the remote investigator can recover the semantic information (e.g., both user-level processes and kernel modules) from the acquired content. When the remote server receives the acquired content, the OS on the target machine has resumed its operation. Thus, the current OS context is different from that when the memory content is acquired.

1) Acquiring Memory: A commercial PCI network card is used to acquire the physical memory of the target machine. PCI network cards can read the physical memory through DMA and then send it out as network packets. However, one problem of using the commercial network cards is that they need a driver. Since the drivers normally reside in OS, if the OS has been compromised by malware, the driver may be compromised too and cannot be trusted. Some researchers use a customized PCI card to read the physical memory [4], which does not need a driver and can overcome the previous problem. The drawback is that the cost for the customized hardware may be high and hard to deploy. We propose to use the commercial network cards without worrying about its drivers. To solve the driver problem, we put the drivers of the network cards into the SMM. Since SMM code is trusted and locked, the driver is not threatened by the malware.

2) Translating Memory: The PCI network card can only acquire the physical memory of the target machine. To understand the physical memory, the system must translate it to virtual memory used by the OS and find out the semantics of the memory. For that purpose, two CPU registers are critical: IDTR and CR3. IDTR points to the current IDT, and CR3 points to the base address of the current page table.

CPU registers can be obtained either through a software based method or a firmware based method. We choose firmware (BIOS) based method, because it is more reliable. We use SMM code, which is a part of the BIOS, to read the CPU registers. As we mentioned in Section II, SMM can obtain the CPU registers used by the OS running in the protected mode, because the hardware automatically saves them before switching to SMM.

3) Challenges: One challenge for off-line investigation is how to perform it reliably on machines with large amount of memories. For example, current workstations or servers may have 4GB, 8GB or even higher amount of physical memory.

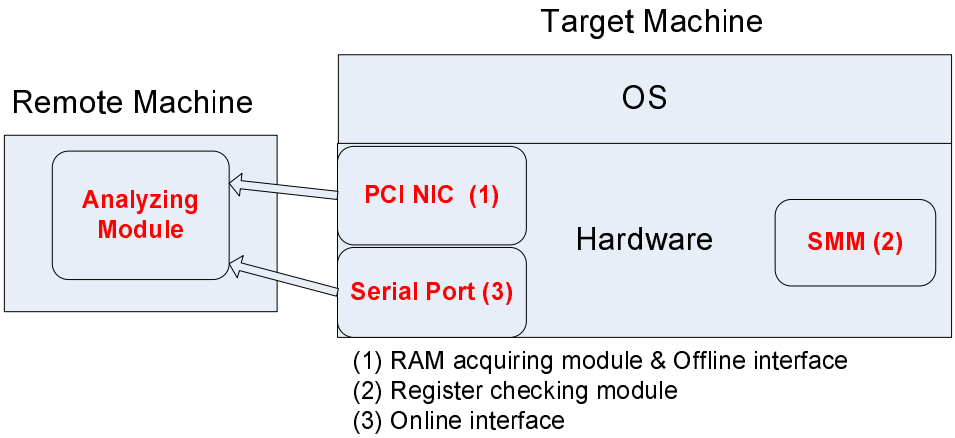

Fig. 1. The architecture of the memory forensics tool.

If we use a 1Gbps network card, then it takes about at least 32 seconds to acquire the 4GB memory, and 64 seconds for 8GB memory. Serial port is not suitable for transmitting the large off-line data due to its low date rate.

There are two ways to acquire the memory: record the whole memory in one operation; or record only a portion of the memory, resume the OS, and then repeat the process until all the memory is recorded. The first method guarantees the memory content to be consistent because it freezes the OS during the process. However, the OS may hang if it is frozen for a relatively long time, partly because too many hardware interrupts are lost when frozen. The second method does not interfere too much to the OS, but the memory acquired during a relative long period may not be consistent. To improve the first method, we could use advanced network cards, such as 10Gbps cards. To improve the second method, we could only capture the difference between the two snapshots of the memory. This will dramatically reduce the amount of data to be recorded. To record the difference, we could mark all the memory pages as read-only, and then every write to the memory will trigger an exception. In the exception handler, the OS records the memory pages to be modified and sends this information to the SMM. This method introduces some performance impacts and modifications to the OS. Also, the exception handler should be protected. SMM can be used to watch the integrity of the exception handler.

B. On-line Investigation

Although the off-line investigation can get the complete memory view of the target machine for a given time, some memory content may not be necessary for the investigation. For example, if the investigator only wants to check the OS kernel, the memory occupied by the user level programs does not need to be recorded or transmitted. Moreover, if the investigator only wants to check a few variables, most of the memory dump will be unnecessary. In this situation, it is more convenient and efficient to use on-line investigation.

In an on-line investigation, the investigator stops the machine, checks a few memory locations or specific registers, and then resumes the machine. We use SMM to stop the machine and get the memory contents. Instead of sending all the memory to the remote machine, SMM just waits for the command from the investigator at the remote server. Only a small number (a few bytes) of memory content will be sent out when it is requested by the investigator through online analysis interface. Since this process is similar to a debugging process, we plan to implement a GDB stub and gdbserver in SMM of the target machine and connect it with a GDB debugger on the remote machine. Gdbserver supports both TCP connection and a serial line connection. For our system, we plan to use serial connection because of simplicity.

1) Triggering SMM: The first step for online investigation is to trigger SMM to freeze the machine. This can be done in several ways. For example, a program or a kernel module can write to port $0xB2$ on an Intel based motherboard to trigger SMM. The PCI network card can be used to trigger SMM too. From the PCI3.0 specification [11], a PCI card can either trigger a normal interrupt or a System Management Interrupt (SMI) that will enter SMM. Another way to trigger SMM is to use a hardware timer to trigger SMM periodically. In this case, the SMM code can ask the remote machine whether it is requested to stop the machine. If not, the code will exit SMM; otherwise, the code will stop the machine and wait for commands from the remote server.

For server machines, another mechanism, IPMI, can also trigger SMI. IPMI stands for the Intelligent Platform Management Interface [12], which is a standardized computer system interface used by system administrators to manage a computer system and monitor its operation. Intel leads the development of IPMI and more than two hundreds computer system vendors support it. IPMI relies on baseboard management controller (BMC) to manage the interface between system management software and platform hardware. BMC is a specialized microcontroller embedded on the motherboard of a computer and is able to trigger SMI. Therefore, IPMI and BMC can be used to trigger SMI remotely[13].

2) Gdbserver and GDB stub: Gdbserver is a control program for Unix-like systems, which allows a user to connect

her program with a remote GDB without linking in the usual debugging stub [14]. The debugging stub, sometimes referred to as GDB stub, is also required to support remote debugging. GDB and gdbserver can communicate with each other using the standard GDB remote serial protocol.

We plan to implement gdbserver and GDB stub in the SMM. When an debugging exception happens, the exception handler will invoke SMM first and then the gdbserver in SMM will communicate with the GDB running on the remote machine. One challenge is how to integrate SMM with gdbserver. Normally, SMM code is one part of the BIOS and written in assembly language. For some old machines, the SMM is not locked by the BIOS so that a third party developer can write her own code and load it into the SMM. But for new machines, the SMM is typically locked by the BIOS. To solve this problem, we use coreboot.

Coreboot [15] (formerly known as LinuxBIOS) is an open source project aimed at replacing the proprietary BIOS (firmware) in majority today's computers. It performs a little bit of hardware initialization and then executes a so-called payload, such as SeaBIOS [16]. Coreboot is a modern version of the firmware, because it switches to protected mode in a very early stage and is written mostly in C language. Coreboot also initializes the serial port to output some debug messages. For our purpose, we could modify the coreboot source code, find the SMI handler and implement the GDB remote serial protocol (RSP) there.

The GDB RSP provides a high level protocol specification allowing GDB to connect to any target remotely. If a target implements the server side of the RSP protocol, the debugger will be able to connect remotely to that target. The protocol supports a wide range of connection types: direct serial devices, UDP/IP, TCP/IP and POSIX pipes.

The RSP is a simple, packet based scheme. The format for each packet is as follows:

```
$ <data> # CSUM1 CSUM2
```

The packet starts with $ symbol, then the actual data, and then # symbol and then two hex bits for checksum. GDB provides many commands for debugging. In our prototype, we plan to first implement the minimum commands that are critical for the live forensic applications. Some commands are:

1) 'g'. 'g' is used to read the content of all the CPU registers. When the GDB server receives this command, it returns the register contents to the client.
2) 'G'. 'G' is used to write the content of the CPU registers. Values to be written are in the same packet with 'G' command. It requires a reply such as 'ACK' or 'NAK'.
3) 'm'. 'm' command reads the content of a specific memory. When received, the GDB server returns the content to the client.
4) 'M'. 'M' command writes the content to a memory address. Values to be written are in the same packet with 'M' command. It requires a reply such as 'ACK' or 'NAK'.
5) 'c'. 'c' command resumes the execution. It can be transmitted with or without an address. If an address is specified, then resume the execution from that address. Otherwise, resume from the current address.

One challenge is that the debugger cannot freeze the target OS in SMM mode too long; otherwise, the target machine may hang. One solution is to quit the SMM mode automatically after a predetermined time period. Another solution is to find out the reason that the OS hangs and try to modify the OS (such as increase the buffer) to support the long time OS frozen situation.

IV. Related work

Traditional digital forensics were designed to retrieve and process all non-volatile machine-derived evidence in an unchanging state, while live digital forensic techniques seek to take a snapshot of the state of the computer similar to a photograph of the scene of the crime [5].

The above "live" forensics definition gave rise to a plethora of recent research on how to analyze the raw memory dumps [17], [18], [3], [4], [5], [19]. Recent work attempts to leverage multiple memory dumps using a tool called "CMAT". CMAT parses a memory dump to find active, inactive and hidden processes as well as system registry information [20]

Complementing the existing semantic approaches, our work focuses on the process of acquiring the system state and thus, it is an enabling technology that enhances the value of existing memory analysis mechanisms.

Previous researchers use hardware-assisted techniques to acquire memory: a special-purpose PCI device can be used either for forensic purpose [4] or for rootkit detection [7], [18]. These devices either need a driver [18] (which is not protected), or need to support stand-alone mode [7] (which may not be supported by many commercial network cards). HyperCheck [21] is close to our work. HyperCheck also uses a PCI network card and SMM to obtain a full view of the target machine. One difference is that HyperCheck is used for integrity monitor while our system is used for forensic analysis. Moreover, we try to solve the problem of how to acquire a large amount of memory, which is not mentioned in the HyperCheck. We also target at providing an on-line forensic analysis by adopting debugging techniques.

Another related work is HyperGuard [22]. Rutkowska *et al.* suggests using SMM of the x86 CPU to monitor the integrity of the hypervisors. Besides using SMM, we also use a PCI network card to perform the analysis of the state snapshot. The goal of our work is different from HyperGuard.

Flicker [23] uses a TPM based method to provide a minimum Trusted Code Base (TCB), which can be used to acquire and report the memory contents to a remote machine. Flicker requires advanced hardware features such as Dynamic Root of Trust Measurement (DRTM) and late launch. In contrast, we only need the static Platform Configuration Registers (PCRs) to secure the booting process. To reduce the overhead of Flicker, TrustVisor [24] has a small footprint hypervisor to perform some cryptography operations. However, TrustVisor is not designed for digital forensic.

A number of recent work has gone towards using SMM to generate efficient rootkits [25], [26], [27], [28]. These rootkits can be used either to get root privilege or as a key-stroke loggers. Our tools use SMM for forensic purpose and are promising to detect those rootkits.

V. Conclusion and Future work

Due to the increased sophistication of malware and the complexity of the software, being able to perform live digital forensics on commodity hardware has become a necessity. An even more desirable feature is to be able to preserve and "step-through" the execution of applications in a stealthy and unobtrusive manner that goes beyond mere semantic analysis of a dump of the memory contents and CPU registers.

In this paper, we discuss our approach to address the problem of acquiring multiple, continuous "snapshots" of the live state of a commodity machine using a network card and CPU SMM. The goal of our system is to perform such operations without disrupting or informing the underlying code of the presence of the inspection system. To that end, we propose a new framework for volatile memory digital forensics that leverages an interactive state analysis component that offers a forensics investigator with a means to access the suspended state of the machine. Additionally, we plan to integrate GDB remote debugging with SMM for online forensics. We believe that this will empower the analyst to perform a much more meaningful analysis of the running programs and provide a more reliable solution than general kernel and debugger-based forensics. This is mainly due to the fact that the CPU SMM-mode depends only on BIOS functionality and once invoked, cannot be bypassed or subverted. As a future work, we plan to fully implement the proposed system and test its performance under different conditions and operating system.

VI. Acknowledgements

This work was supported by the United States Air Force Research Laboratory (AFRL) through Contract FA8650-10-C-7024. Opinions, findings, conclusions and recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the US Government.

References

[1] CAIDA, "Caida analysis of code-red, http://www.caida.org/research/security/code-red/." [Online]. Available: http://www.caida.org/research/security/code-red/

[2] X-Force, "Sql slammer worm propagation. http://xforce.iss.net/xforce/xfdb/11153." [Online]. Available: http://xforce.iss.net/xforce/xfdb/11153

[3] G. G. Richard, III and V. Roussev, "Next-generation digital forensics," *Commun. ACM*, vol. 49, pp. 76–80, February 2006. [Online]. Available: http://doi.acm.org/10.1145/1113034.1113074

[4] B. D. Carrier and J. Grand, "A hardware-based memory acquisition procedure for digital investigations," *Digital Investigation*, vol. 1, no. 1, pp. 50 – 60, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/B7CW4-4BMXXJS-B/2/74a97eb4c390ef6a2a2f8111c3a59aa5

[5] F. Adelstein, "Live forensics: diagnosing your system without killing it first," *Commun. ACM*, vol. 49, pp. 63–66, February 2006. [Online]. Available: http://doi.acm.org/10.1145/1113034.1113070

[6] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold boot attacks on encryption keys," in *Proceedings of the 17th conference on Security symposium*. Berkeley, CA, USA: USENIX Association, 2008, pp. 45–60. [Online]. Available: http://portal.acm.org/citation.cfm?id=1496711.1496715

[7] 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.

[8] R. Hiremane, "Intel® Virtualization Technology for Directed I/O (Intel® VT-d)," *Technology© Intel Magazine*, vol. 4, no. 10, 2007.

[9] AMD, "IOMMU Architectural Specification. Advanced Micro Devices, Inc."

[10] Intel Corp., "Intel® 64 and IA-32 Architectures Software Developer's Manual," June 2010.

[11] PCI-SIG, "PCI Local Bus Specification Revision 3.0," September 2010. [Online]. Available: http://www.pcisig.com/specifications/conventional/

[12] Intel, HP, NEC and Dell, "IPMI intelligent platform management interface specification second generation v2.0. ."

[13] A. Azab, P. Ning, Z. Wang, X. Jiang, and X. Zhang, "HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity," in *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS 2010)*, 2010.

[14] "GDB manual, http://www.gnu.org/software/gdb/documentation/." [Online]. Available: http://www.gnu.org/software/gdb/documentation/

[15] "coreboot http://coreboot.org/." [Online]. Available: http://aufs.sourceforge.net/

[16] "Seabios, http://www.coreboot.org/seabios."

[17] N. L. Petroni, J. Aaron, W. Timothy, F. William, and A. Arbaugh, "Fatkit: A framework for the extraction and analysis of digital forensic data from volatile system memory," *Digital Investigation*, vol. 3, 2006.

[18] 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.

[19] M. Carbone, W. Cui, L. Lu, W. Lee, M. Peinado, and X. Jiang, "Mapping kernel objects to enable systematic integrity checking," in *Proceedings of the 16th ACM conference on Computer and communications security*. ACM, 2009, pp. 555–565.

[20] F. Law, P. Chan, S.-M. Yiu, B. Tang, P. Lai, K.-P. Chow, R. Ieong, M. Kwan, W.-K. Hon, and L. Hui, "Identifying volatile data from multiple memory dumps in live forensics," in *Advances in Digital Forensics VI*, ser. IFIP Advances in Information and Communication Technology, K.-P. Chow and S. Shenoi, Eds. Springer Boston, 2010, vol. 337, pp. 185–194.

[21] J. Wang, A. Stavrou, and A. K. Ghosh, "HyperCheck: A Hardware-Assisted Integrity Monitor," in *Proceedings of the 13th International Symposium on Recent Advances in Intrusion Detection (RAID 2010)*, 2010.

[22] J. Rutkowska and R. Wojtczuk, "Preventing and detecting Xen hypervisor subversions," *Blackhat Briefings USA*, 2008.

[23] J. McCune, B. Parno, A. Perrig, M. Reiter, and H. Isozaki, "Flicker: An execution infrastructure for TCB minimization," in *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*. ACM, 2008, pp. 315–328.

[24] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. Gligor, and A. Perrig, "TrustVisor: Efficient TCB reduction and attestation," in *Proceedings of the IEEE Symposium on Security and Privacy*, May 2010.

[25] F. Wecherowski and core collapse, "A Real SMM Rootkit: Reversing and Hooking BIOS SMI Handlers," *Phrack Magazine*, 2009.

[26] BSDaemon, coideloko, and D0nAnd0n, "System Management Mode Hack: Using SMM for "Other Purposes"," *Phrack Magazine*, 2008.

[27] S. Embleton, S. Sparks, and C. Zou, "SMM rootkits: a new breed of OS independent malware," in *Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*. ACM, 2008, p. 11.

[28] L. Duflot, D. Etiemble, and O. Grumelard, "Using CPU System Management Mode to Circumvent Operating System Security Functions," in *Proceedings of the 7th CanSecWest conference*. Citeseer, 2001.