Hardware Virtualization Based Security Solution for Embedded Systems

Sandor Lukacs[*#], Andrei V. Lutas[*#], Dan H. Lutas[*#], Gheorghe Sebestyen[#]
[*]Bitdefender, Cluj-Napoca, Romania
[#]Technical University, Cluj-Napoca, Romania
{slukacs, alutas, dlutas}@bitdefender.com gheorghe.sebestyen@cs.utcluj.ro

Abstract—**We describe the implementation and the evaluation of a hypervisor level, hardware-enforced security solution suitable for the latest embedded platforms. Our solution is based on thin layer bare-metal hypervisor, a memory introspection engine and is validated on Silvermont microarchitecture based Intel x86 processors, running Windows. The approach is well suited to enhance the security of many POS and industrial embedded devices. We also present various kinds of attacks our solution defends against, and several remaining limitations.**

Keywords—embedded, hypervisor, virtualization, Silvermont, security, malware, introspection

I. INTRODUCTION

Security is a critical aspect when it comes to any computing device, be it a personal computer, a smart phone or an embedded device. In the past years, the embedded systems realm has been the target of several complex, well planned attacks. The most notorious of them, Stuxnet [1], has been discovered on several SCADA systems within a nuclear power plant in Iran. This worm was unlike any other seen before: it was infecting Siemens industrial control systems, via 4 zero-day vulnerabilities inside the Microsoft Windows operating system. Stuxnet appeared to be specifically created for this purpose, and it had the capability of stealing data and issuing control commands to the control systems. This is, perhaps, the most notable case of embedded systems attack. More recently, malware present on the POS devices of several Target stores in the United States helped on compromising more than 40 million credit-cards [2]. This is a typical case of cyber-attack having financial theft as the main motivational factor. There are many other cases where simpler devices, such as a printer, a network router or even a tablet is being compromised by malware [3]. It's clear that embedded security becomes an important aspect, especially in these days, when everyday-life devices, like cars, airplanes, POS and ATM machines, health-care devices or military equipment are based on embedded processors and systems [4]. Moreover, while conventional attacks may typically lead to identity theft, data exfiltration or financial loss, a successful attack on an embedded system may have substantially greater impact, including failure of critical industrial control systems or loss of human lives.

In the last twenty years we have seen the large scale adoption of personal computers – most of them based on x86 architecture – paired with a growing number of sophisticated malware attacks [5]. Windows XP is being ran on a large number of client computers, and since the embedded version features many times 10-15 years or more of lifetime [6], it is expected that many systems will keep running it well beyond the date the support for this version of Windows is dropped. It will open a whole new world of possible cyber-attacks [7]. Together with the possibility of an increase in the number of x86 embedded processors, we see again the most common combination of vulnerable systems: Microsoft Windows running on an x86 CPU. The need of an embedded security solution capable of leveraging the latest technologies becomes more and more obvious.

While the industry and academia both tried to continuously create various new malware prevention and detection solutions, it is fairly clear that all current conventional security solutions can be easily bypassed by malware [8], [9]. Many advocate that a paradigm shift is needed, implying migration from software-only to hardware-enforced security solutions – many of which implies hardware virtualization [10]–[12]. We fear that history tends to repeat itself: the massive scale adoption of mobile devices [13], the incoming era of internet-of-things – Intel forecasts 200 billion internet connected devices till 2020 [14] – will create huge security issues from which cybercriminals will certainly profit.

The embedded market is also rapidly growing [15], and there are voices [16] that advocate for the adoption of virtualization in embedded systems also. Even though x86 architecture is highly prevalent in embedded systems [17], [18], till the recent launch of Intel's low-power Atom series of microprocessors, based on Silvermont microarchitecture [19] there was no possibility to adapt the latest PC class security solutions, based on hardware virtualization, to low power SoC embedded devices. While ARM low-power CPUs supporting hardware accelerated virtualization have been introduced already in 2012 [20], there are many areas, including ATM devices [7], where Intel x86 and Microsoft Windows are overwhelmingly dominating the embedded market.

In this article, we will present our view and our proposed solution towards securing embedded operating systems and applications using a bare-metal hypervisor and a memory introspection engine. We will present the current progress, the vision, the limitations and the future development of such a security solution, validating our results on the latest embedded Intel Atom processors and Windows operating systems. While we strongly believe that employing hypervisor level security in embedded systems can significantly raise the cost of an attack, at the end of the paper, we will point out several remaining attack vectors and future research areas.

II. Security and Isolation

A. *Privilege levels & isolation*

The concept of running software with different privilege levels and isolating processes and tasks into different security domains is well known and established [21]. Traditionally, operating systems implement isolation by separating user-mode processes from the operating system kernel. The first x86 CPU that implemented protected mode was the Intel 80286 [22]. This CPU featured a protection mechanism based on privilege rings: ring 0, ring 1, ring 2 and ring 3. From this, mainstream operating systems used only two: ring 3 for user mode processes and ring 0 for operating system kernel. This provides the basics of isolation between the operating system code and data (the *kernel*) and the user applications. Later, the Intel 80386 CPU introduced a new addition: paging. This provides the basics of address space isolation: code and data residing at the same privilege level inside an address space can't access code and data that resides in a different address space, even though it has the same privilege level (ring). The paging mechanism forms the basics of virtual memory and inter-process isolation [5].

It is important to point out, that the protection rings and address space isolation are controlled by CPU and MMU mechanisms that are hardware enforced, thus, conceptually should have been able to provide adequate isolation and protection. One of the key reason why this setup fails to provide adequate protection is because widely used operating systems, for performance reasons, all tend to be built upon monolithic approaches: the OS kernel, the drivers and many other modules share a common CPU privilege level and the same common address space. The operating system kernel, the drivers and various modules however represent a very large attack surface, where finding an exploitable vulnerability due to software coding error is a widespread practice [23].

There are also other approaches to implement isolation. Software sandboxes are used by word processors, PDF readers or browsers for example [24], trying to impose stronger isolation between potentially vulnerable processes and the operating system kernel. In yet another approach [25] Porter et al proposes a library OS to reduce the attack surface of an isolation. While this approach is certainly promising, it requires a significant change of the operating system interfaces, being infeasible to be done for existing OSes. Several additions have been made lately also to the paging mechanism, improving the security features of the CPU. These include the execute disable bit (NX) [22], supervisory mode execution prevention (SMEP) and supervisory mode access prevention (SMAP).

All of these approaches provide a somewhat increased level of security and better resistance against exploits and various attacks. However it is fundamental to note, that all of them are controlled by the operating system kernel, from the privilege level and from within the address space of the kernel – thus, their success is limited again by the large attack surface exposed by the kernel [26].

B. *Hardware level virtualization*

Intel introduced VT-x hardware supported virtualization in 2006 [22], providing the basics of hardware assisted and enforced virtualization. VT-x technology implements an additional privilege ring (VMX root mode, or ring -1), which resides logically below the operating system. This opens a lot of opportunities for creating security solutions that lie below the operating system. VT-x offers a hypervisor the capability to be executed at a greater privilege level than the operating system kernel. The recent processors support also hardware accelerated second level address translation (known as Extended Page Tables, EPT on Intel x86), which provides a hypervisor with the ability to isolate the memory pages of the operating system and of various processes into several different protection domains. It is important to point out, that the isolation is configured and controller by the hypervisor running below the operating system and is enforced by the hardware (e.g. both the configuration and enforcement of isolation are out of the reach of the OS kernel, and out of the reach of a kernel rootkit, that already successfully attacked the OS kernel).

On low level, virtualization technologies like VT-x have a *trap & emulate* architecture: certain instructions will cause o fault-like event (called VM-exit in the case on Intel VT-x), which will be trapped and handled by the virtual machine monitor. When the extended page tables mechanism is in use, the virtual addresses will translate, via the guest paging structures, to a guest physical address (or virtualized physical address). The hardware will then translate this guest physical address to a machine physical address, or real physical address, using the Extended Page Tables. Inside the EPT, access rights bits decide whether data inside that page can be read, written or executed.

This opens several opportunities for security software. From an isolation standpoint, the virtual machine monitor resides in a different address space, inaccessible to the guest operating systems. It's worth mentioning, that any security system is as secure as its weakest link, therefore, a misconfigured hypervisor would most likely provide a very limited increase in security.

Intel VT-x (virtualization technology), Intel EPT (extended page tables), Intel VT-d (I/O virtualization), Intel TXT (trusted execution technology) are all technologies capable of supporting a security solution to provide true separation from a hostile process or operating system. Many of those technologies are present in recent mainstream consumer processors, but up until October 2013, they were not present in low-power architectures, specific to embedded architectures. The latest x86 microarchitecture, named Silvermont [19][27], is the first low power microarchitecture, well suited for embedded and industrial applications that features also Intel VT-x and EPT hardware virtualization technologies, thus opening several new possibilities towards porting existing field proven advanced security solutions from PCs to embedded systems.

III. Our Proposed Solution

A. High-level approach

We propose a solution based on a bare-metal, type 1, thin-layer, security-oriented hypervisor, capable of running on embedded systems with a low overhead, and thus implicitly in a power-friendly manner. Considering the fact, that full blown hypervisors that support multiple VMs represent themselves a considerable attack surface [28], [29], there is a strong need to employ a thin layer hypervisor, specially tailored hypervisor that incorporate only the security features needed. Thin layer hypervisors have a much reduced footprint also because they do not need to incorporate hardware device virtualization. This way, we choose to support performance and security, but limit ourselves to support only single VM scenarios.

As seen in Figure 1, the hypervisor integrates a Memory Introspection Engine (MIE) [30], capable of analyzing, identifying and protecting the running guest operating system. The introspection engine is capable of providing two main types of protection:

- *Kernel integrity protection* – including protection of the images of the kernel and critical drivers against advanced kernel mode malware and exploits,
- *User mode processes protection* – including, but not limited to exploit prevention and enforcing various per-process security policies.

A particularly important capability of the MIE is that is dynamically adaptable by updating its security policies (e.g. whether if it permitted or denied the injection of a given code module from a user mode process into another). A key requirement was for the MIE module to avoid relying on any in-guest agent, being able to identify all in-guest structures by parsing raw memory pages and identifying those structures using invariant signatures. Besides being very secure, this approach provides the advantage that the hypervisor and any MIE engine based security solution can be easily and completely independently updated from the in-guest operating system or any in-guest security solution.

B. Architecture and functionallity

The Memory Introspection Engine (MIE) is capable of providing two levels of protection. First of all, it provides an integrity-check mechanism, using which critical structures can be periodically validated against malicious modifications. This mechanism is similar to the Windows x64 PatchGuard, but unlike the latter, it is not vulnerable to attacks, since it resides outside the guest.

The protection types that the MIE offers fall into two main categories. First of all, the kernel integrity protection will prevent a kernel rootkit from altering critical areas of the operating system. To achieve this, we protect critical areas including, among others, the core kernel image (*ntoskrnl*), the hardware abstraction layer (*hal*), several critical core drivers (such as the disk and network drivers), third party drivers (such as those belonging to an in-guest security solution), driver objects & fast I/O dispatch tables (both of which contain critical pointers to I/O routines), page tables, the Interrupt Descriptor Table and the Global Descriptor Table used by the operating system. The MIE is capable of scanning the guest virtual machine's physical memory range and identifying all these structures, frequently targeted by rootkits and kernel exploit attacks. In our implementation, a key target was to be able to do all those identifications without relying on any in-guest component, thus being able to bridge the semantic gap between the rich view of the structures as seen by the guest operating system and the raw view seen by the introspection engine. We achieve this, by using invariant data structures and code signatures, according to different operating system versions. The detection of the initial kernel structures is done by analyzing the system call Model Specific Registers and the entries inside the Interrupt Descriptor Table, all of which point to operating system specific handling routines. Many of the internal structures are undocumented, so the exact details could be obtained only by using reverse engineering.

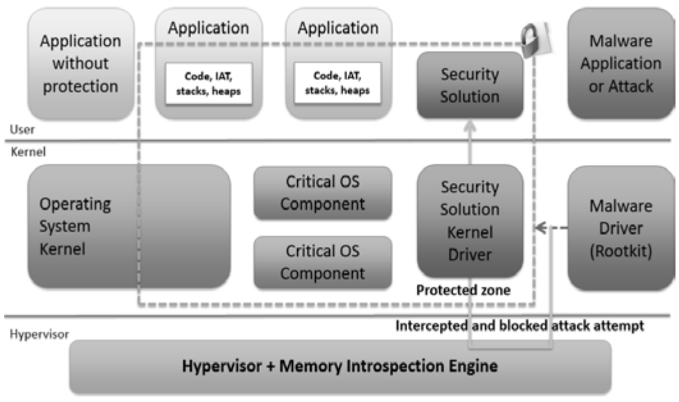

Fig 1. Hardware virtualization and MIE engine based security solution

The second protection type offered by the introspection engine is the process protection. Several user-mode processes, deemed *critical*, can be protected from within the hypervisor, by preventing code and data injections inside of them, by blocking various hooking attempts, and by blocking possible exploitation attempts via buffer overflows. In order to achieve this, certain internal Windows API functions are intercepted at the hypervisor level in order to detect events such as process and thread creation or termination. Each of these events will then be used to setup & activate, adapt or terminate protection on a per-process and per-thread basis. Among others, the protected areas of processes include stacks and heaps, which are protected against code execution (and thus buffer-overflow exploitation), several critical system libraries (such as *ntdll* and *kernel32*, which are protected against hooking). The main module is protected against polymorphic code execution and the entire process image is protected against multiple types of code and data injection attempts.

In order to prevent code execution or malicious writes, the MIE will mark all the pages that contain critical kernel or user structures, inside the EPT, according to their functional semantics as being either non-executable (e.g. stacks and heaps) or non-writable (e.g. code pages). Each time code is fetched from within a stack or heap page, or data is being written inside a protected structure, a VM exit will be generated by the CPU, transferring control from the guest VM to the below-OS introspection engine. The MIE then will analyze the violation, and decide whether it is legitimate or not, based on both various contextual information (e.g. such as previous and new value of a memory location) and exception signatures or built into MIE heuristics logic. This way, the number of false-alarms is greatly reduced, and new false-alarms can be easily solved by adding an exception signature. Code injections are detected by intercepting certain internal OS functions.

The Memory Introspection Engine supports all Microsoft Windows operating system versions, beginning with Windows Vista and up until Windows 8.1 on 32 and 64 bit platforms.

IV. Evaluation

A. *Evaluation of protection against various kinds of attacks*

To empirically evaluate the attack resistance of our setup, in a first stage we used various well known kernel rootkits trying to attack a protected system. For example, the System Service Dispatch Table (SSDT) is a critical kernel table which contains function pointers to various system-call handlers. The MIE is capable of protecting it and preventing specific attacks generated by rootkits that intend to hook certain entries inside this table. One notorious example of such rootkit is Jadtre, which is successfully blocked. There are thousands of rootkits that employ this technique in order to achieve stealthiness & persistence. Another type of attack the MIE protects against is inline hooking inside critical kernel components. The *ntoskrnl* kernel image is the target of several rootkits, such as Bagle, which attempts to establish inline hooks on certain kernel functions, in order to hide malicious processes. By protecting the kernel image and the critical kernel components, such as *hal*, the MIE is capable of easily blocking these attacks. The TDL [31] rootkit became famous, due to its infection technique. One important step the rootkit has to employ is hooking certain function pointers inside a specific driver object structure. However, since the MIE is able to identify and protect the driver objects, the TDL rootkit is blocked as it attempts to establish low-level hooks inside the kernel.

On a separate set of attack attempts we tried to validate the effectiveness of the user-mode process protection. Examples of blocked attacks include exploits (like CVE-2010-3333, which is a stack overflow vulnerability in Microsoft Word), code injections (technique employed by most of today's malware, for example, ZBot, which tries to inject malicious code into legitimate processes, in order to trick security solutions), malicious polymorphic code (Sality and Virtob are two file infectors that append polymorphic code to otherwise legitimate applications – this polymorphic code is detected at runtime by the MIE) and hooks, which include both inline hooks inside system libraries such as *ntdll* or *kernel32*, and export/import table hooking of these important libraries (examples of such attacks include most of today's malware – ZBot, Sality, Virtob among others).

B. *Evaluation of performance impact (overhead)*

The performance tests were done on a custom built system with off-the-shelf components, including a Gigabyte J1800N-D2H mainboard having onboard an Intel Celeron J1800 CPU with 2 cores at 2.41-2.58 GHz, 2 x 1 GB 1333 MHz DDR3 CL9 memory and a SATA2 5400 rpm hard disk. The selected hardware configuration is representative for embedded setups, like point-of-sale systems. The system was running Windows 8.1 x64. In order to obtain consistent performance measurements, we disabled a set of well-known background services, including Windows Defender, Windows Search, Windows Update, SuperFetch and disk defragmenter.

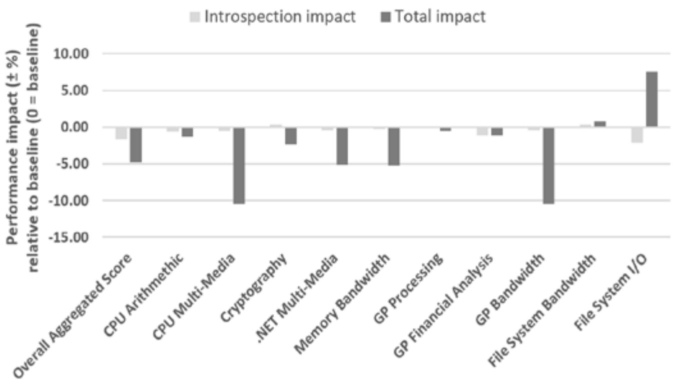

Fig 2. Performance overhead for the J1800 CPU based system

We decided to run various benchmarks representative for the overall performance of the test systems, using SiSoftware Sandra Lite 2014 SP2 (v20.28). The benchmarks comprise many complex scenarios, covering areas such as cryptography, multi-media, file I/O operations, general purpose (CPU+GPU) processing or financial analysis also.

First of all, a baseline was established by running each test for three times and averaging the results. Next, we repeated the measurements with our hypervisor running, but without having the MIE active. Lastly, we ran the measurements with both the hypervisor and MIE protecting the system. We determined the *Introspection Impact* (light bars) comparing the scores from the last scenario, where both the hypervisor and MIE are enabled with the second scenario measurements, where only the hypervisor was running. We calculated the *Total Impact* (dark bars) comparing the scores from the last scenario, where both the hypervisor and MIE are enabled with the baseline results. As the various benchmarks are measured in different units, we choose to normalize the values relative to the baseline results and represent the differences in percent.

Figure 2 presents the impact measured against the baseline. The *Overall Aggregated Score* represents a summarized score that includes all the various benchmarks that follows it. The overall result is clear: the memory introspection technology is very affordable from performance point of view, bringing on average around -2% impact, and in the case of some workloads the impact being almost unmeasurable. A greater but still reasonable impact can be attributed to the virtualization layer, which at two benchmarks (*CPU Multi-Media* and *GP Bandwidth*) generates an up to -10% impact. The results support our initial premise that hardware enforced security solutions can be used with low impact on embedded systems.

While on average, running the same benchmarks multiple times generated consistent and straightforward results, we also obtained some unexpected, but repeatedly reproduced data, for which we are still investigating the exact cause. While the small positive impact – meaning a performance improvement – measured for *File System Bandwidth* could be explained by the inaccuracy of the measurement method (the average measured improvement being less than 0.75%), the average 9.8% improvement for the *File System I/O* benchmark is unrealistic. We suspect this to be attributable to some yet to be found interaction or incompatibilities between the hypervisor and the benchmarking software we used. Another theoretical possibility is, that this might be caused by some sort of interfering with the data or cache access patterns. We do not believe this speedup to be genuine and will further investigate it.

V. Remaining Limitations

Although initial steps have been taken by Intel towards implementing virtualization technologies on low-power SoC systems, they still lack advanced security features such as I/O virtualization (Intel VT-d) or Trusted Execution Technology (TXT). Therefore, advanced targeted attacks could, at least theoretically, disable our security solution or even inject malicious code in it that would gain the same privilege as the hypervisor itself – for example using DMA attacks [32]. However, recent advances in the embedded processors give us reason to believe that even the more advanced security solutions, such as Intel SGX (Software Guard eXtensions) [36] will also be ported to embedded processors, opening new opportunities for security software.

Vulnerabilities in the CPU or in hardware virtualization technologies can carry significant security impact, leading to privilege escalation and even bypassing of critical security technologies. Software attacks on Intel VT-d [35] and Intel TXT [36] were used to demonstrate guest-to-host escapes. Also, vulnerabilities that allow malicious code to crash the system [37] or perform operating system privilege escalation [38] or even hypervisor compromises were discovered [29].

System firmware, such as SMM or UEFI, can also contain vulnerabilities leading to full system compromise, as demonstrated by Wojtczuk [39] and Duflot [40]. Mitigating such attacks can be done by virtualizing SMM, however this is only supported on AMD platforms, at least for now. Device firmware (e.g. network cards with Intel AMT) have also been shown to be exploitable, allowing remote and undetectable compromise of the system. Such attacks were presented by Tereshkin [41] and Duflot [42]. Mitigations for this class of attacks are isolating the network card inside a separate network dedicated virtual machine, or not deploying network cards using Intel vPRO/AMT (and ensuring proper updating of device firmware from trusted sources).

As demonstrated by Skorobogatov [43] and Brossard [44], possibility of hardware backdoors should also be considered. While the deployment of such backdoors in commercial applications is unlikely to be affordable for average cybercriminals, given the recent NSA related leaks [45]–[47], we stress that maximum precaution is more than needed while considering government capabilities. No generic and efficient mitigation measures are available for now for this kind of attacks.

This section was not meant to be exhaustive, but rather only exemplificative. More attacks to consider have been briefly described by Lutas et al [48].

VI. Conclusions

A. Achievements

Our work demonstrated the practical feasibility of running a thin-layer hypervisor based memory introspection engine on embedded systems. We implemented protection mechanisms not only for kernel mode structures, but also for user-mode processes, thus being able to better protect the integrity of various applications against many kind of attacks.

We validated that our proposal works on real-life systems, based on the latest low-power Intel x86 Atom processors and Microsoft Windows operating systems. We conducted various empirical tests to ensure that the protection mechanisms implemented detects all known kernel rootkit attacks and covers a considerable list of various attack vectors – thus while being impossible to prove, we strongly believe that our solution can raise the cost of future attacks. While not ideal, our performance measurements still clearly proves us that the practical deployability of such solution is feasible.

B. Future Research Directions

There are several next steps we would like to incorporate into our research. We are already working towards validation our approach on other operating systems, such as various flavors of Linux or Android.

On the x86 platform, as soon as more of the hardware security technologies currently found only on PC class systems – like VT-d, TXT or SGX – are introduced also in low power embedded systems, a lot of new opportunities will arise for solving many of the remaining limitations. As an example, input/output virtualization must be addressed as soon as VT-d technology is implemented on these chips in order to properly handle DMA attacks.

We are looking forward to adapt our technology to new architectures also. This year we will likely see the wide scale introduction of 64 bit ARM devices, with Cortex A53 and A57 chips supporting hardware virtualization [49], thus being an excellent platform to adapt to and validate on our research.

References

[1] R. Langner, “To Kill a Centrifuge: A Technical Analysis of What Stuxnet’s Creators Tried to Achieve,” The Langner Group, 2013.

[2] B. Krebs, “A First Look at the Target Intrusion, Malware — Krebs on Security,” 15-Jan-2014. Available: http://krebsonsecurity.com/2014/01/a-first-look-at-the-target-intrusion-malware/. [Accessed: 30-Jan-2014].

[3] A. Ki-Chan and H. Dong-Joo, “Embedded devices, and Antivirus-free safe jideout for malware,” presented at the Defcon, 2010.

[4] “Military Design Trends: Applications Migrating to Intel x86 Architecture,” Kontron, 2010.

[5] “Microsoft Security Intelligence Report, Special Edition, a 10 Year Review,” Microsoft, Feb. 2012.

[6] “Windows Embedded | Product Lifecycles,” 31-Jan-2014. Available: http://www.microsoft.com/windowsembedded/en-us/product-lifecycles.aspx. [Accessed: 30-Jan-2014].

[7] N. Summers, “ATMs Face Deadline to Upgrade from Windows XP - Businessweek,” *BloombergBusinessweek*, 16-Jan-2014. Available: http://www.businessweek.com/articles/2014-01-16/atms-face-deadline-to-upgrade-from-windows-xp#p1. [Accessed: 30-Jan-2014].

[8] M. Christiansen, “Bypassing Malware Defenses,” SANS Institute, May 2010.

[9] R. Beggs, “Failure of Antivirus,” Toronto Area Security Klatch, Oct-2006.

[10] M. Zaharia, S. Katti, C. Grier, V. Paxson, S. Shenker, I. Stoica, and D. Song, “Hypervisors as a Foothold for Personal Computer Security - An Agenda for the Research Community,” University of California, Berkeley, 2012.

[11] G. Heiser, “Hypervisors for Consumer Electronics,” in *Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference*, Piscataway, NJ, USA, 2009, pp. 614–618.

[12] K. Gudeth, M. Pirretti, K. Hoeper, and R. Buskey, “Delivering Secure Applications on Commercial Mobile Devices: The Case for Bare Metal Hypervisors,” in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, New York, NY, USA, 2011, pp. 33–38.

[13] C. Arthur, “Mobile internet devices ‘will outnumber humans this year’ | theguardian.com,” *TheGuardian*, 07-Feb-2013. Available: http://www.theguardian.com/technology/2013/feb/07/mobile-internet-outnumber-people. [Accessed: 30-Jan-2014].

[14] INTEL, “A Guide to the Internet of Things Infographic.”. Available: https://www-ssl.intel.com/content/www/us/en/intelligent-systems/iot/internet-of-things-infographic.html. [Accessed: 30-Jan-2014].

[15] IDC, “IDC: Embedded Systems Market to Double by 2015 | PCWorld,” 09-Sep-2011. Available: http://www.pcworld.com/article/239798/idc_embedded_systems_market_to_double_by_2015.html. [Accessed: 30-Jan-2014].

[16] W. Keegan and A. Subbarao, “Type Zero Hypervisor – the New Frontier in Embedded Virtualization | EECatalog Military & Aerospace,” 14-Aug-2012. Available: http://eecatalog.com/military/2012/08/14/type-zero-hypervisor-%E2%80%93-the-new-frontier-in-embedded-virtualization/. [Accessed: 30-Jan-2014].

[17] VDC, “Embedded x86 Shares Grow from Slow Past,” *embeddedintel.com*, 2009. Available: http://www.embeddedintel.com/market_watch.php?article=1584. [Accessed: 30-Jan-2014].

[18] VDC, “VDC Research Finds ARM Holdings Gaining Market Share in High Growth Processor Applications,” *prweb.com*, 2013. Available: http://www.prweb.com/releases/ARM/Intel/prweb11303199.htm. [Accessed: 30-Jan-2014].

[19] B. Kuttanna, “Technology Insight: Intel Silvermont Microarchitecture,” presented at the Intel IDF, 2013.

[20] A. Shah, “ARM Expects First Cortex-A15 Devices in Late 2012 | PCWorld,” *pcworld.com*, 19-Apr-2011. Available: http://www.pcworld .com/article/225624/article.html#tk.rss_news. [Accessed: 31-Jan-2014].

[21] P. A. Karger and A. J. Herbert, “An Augmented Capability Architecture to Support Lattice Security and Traceability of Access,” *IEEE Symp. Secur. Priv.*, pp. 2–12, 1984.

[22] INTEL, *Intel 64 and IA-32 Architectures Software Developer’s Manual*. Intel, 2013.

[23] MITRE, “Windows Kernel Vulnerabilities,” *MITRE CVE Vulnerability Database*, 30-Jan-2014. Available: http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=Windows+kernel. [Accessed: 30-Jan-2014].

[24] “Invincea FreeSpace for Enterprise Architects,” Invincea, 2013.

[25] D. E. Porter, S. Boyd-Wickizer, J. Howell, R. Olinsky, and G. C. Hunt, “Rethinking the Library OS from the Top Down,” in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, 2011, pp. 291–304.

[26] R. Wojtczuk and R. Kashyap, “The Sandbox Roulette: Are you ready for the gamble?” presented at the BlackHat EU, 2013.

[27] “Leading the Next Industrial Revolution,” Intel, 2013.

[28] J. Couzins, “Attacking the hypervisor | John’s Virtualisation/Security Blog,” 27-Nov-2013. .

[29] N. MacDonald, “Yes, Hypervisors Are Vulnerable,” *Gartner Blog*, 26-Jan-2011. .

[30] T. Garfinkel and M. Rosenblum, “A Virtual Machine Introspection Based Architecture for Intrusion Detection,” in *In Proc. Network and Distributed Systems Security Symposium*, 2003, pp. 191–206.

[31] A. Matrosov and E. Rodionov, “TDL3: The Rootkit of All Evil?” Eset, 2011.

[32] D. Aumaitre and C. Devine, “Subverting Windows 7 x64 Kernel with DMA attacks,” presented at the HackInTheBox, 2010.

[33] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, “Innovative instructions and software model for isolated execution,” *Proc. 2nd Int. Workshop Hardw. Archit. Support Secur. Priv. - HASP 13*, pp. 1–1, 2013.

[34] “Intel Architecture Instruction Set Extensions Programming Reference,” Intel, 2013.

[35] R. Wojtczuk and J. Rutkowska, “Software Attacks on Intel VT-d,” Invisible Things Lab, Apr. 2011.

[36] R. Wojtczuk and J. Rutkowska, “Attacking Intel TXT - paper,” presented at the BlackHat USA, Feb-2009.

[37] Kill3r, “Full Disclosure: Possible VT-x enabled Intel CPU Crash Vulnerability,” 31-Mar-2010. Available: http://seclists.org/fulldisclosure/2010/Mar/550. [Accessed: 30-Jan-2014].

[38] G. Dunlap, “The Intel SYSRET privilege escalation – blog.xen.org,” 13-Jun-2012. .

[39] R. Wojtczuk and J. Rutkowska, “Attacking SMM Memory via Intel CPU Cache Poisoning,” Invisible Things Lab, 2009.

[40] L. Duflot, O. Levillain, B. Morin, and O. Grumelard, “Getting into the SMRAM: SMM Reloaded,” presented at the CanSecWest, 2009.

[41] A. Tereshkin and R. Wojtczuk, “BHUSA09-Tereshkin-Ring3Rootkit-SLIDES.pdf,” BlackHat USA, 2009.

[42] L. Duflot, Y. Perez, and B. Morin, “What if you can’t trust your network card?” *Recent Adv. Intrusion Detect.*, vol. 6961, pp. 378–397, 2010.

[43] S. Skorobogatov and C. Woods, “Breakthrough silicon scanning discovers backdoor in military chip,” *Cryptogr. Hardw. Embed. Syst. Workshop CHES 2012*, no. September, pp. 9–12, 2012.

[44] J. Brossard, “Hardware backdooring is practical,” presented at the BlackHat USA, 2012.

[45] J. Mick, “Tax and Spy: How the NSA Can Hack Any American, Stores Data 15 Years,” *DailyTech*, 31-Dec-2013. Available: http://www.dailytech.com/Tax+and+Spy+How+the+NSA+Can+Hack+Any+American+Stores+Data+15+Years/article34010.htm. [Accessed: 30-Jan-2014].

[46] J. Mick, “Businesses Deny Helping NSA Plant Bugs in Americans’ Gadgets,” *DailyTech*, 01-Jan-2014. Available: http://www.dailytech.com/Businesses+Deny+Helping+NSA+Plant+Bugs+in+Americans+Gadgets/article34022.htm. [Accessed: 30-Jan-2014].

[47] N. McAllister, “How much did NSA pay to put a backdoor in RSA crypto? Try $10m,” *The Register*, 21-Dec-2013. Available: http://www.theregister.co.uk/2013/12/21/nsa_paid_rsa_10_million/. [Accessed: 30-Jan-2014].

[48] D. H. Lutas, S. Lukacs, R. V. Tosa, and A. V. Lutas, “Towards Secure Network Communications With Clients Having Cryptographically Attestable Integrity,” in *Proceedings of the Romanian Academy*, 2013, vol. 14, pp. 338–356.

[49] M. Yam, “AMD Embedded Product 2014 Roadmap Packs ARM, x86 SoC,” *tomshardware.com*, 11-Sep-2013. Available: http://www.tomshardware.com/news/amd-arm-x86-soc-embedded,24196.html. [Accessed: 31-Jan-2014].