Virtual Machine Introspection with Xen on ARM

Tamas K Lengyel, Thomas Kittel and Claudia Eckert
Technische Universität München
{tklengyel | kittel | eckert}@sec.in.tum.de

***Abstract*—In the recent years, virtual machine introspection has become a valuable technique for developing security applications for virtualized environments. With the increasing popularity of the ARM architecture and the recent addition of hardware virtualization extensions there is a growing need for porting existing tools to this new platform. Porting these applications requires proper hypervisor support, which we have been exploring and developing for the upcoming Xen 4.6 release. In this paper we explore using ARM's two-stage paging mechanisms with Xen to enable stealthy, efficient tracing of guest operating systems for security purposes.**

I. Introduction

Over the last decade, significant research and development efforts have been made to create out-of-band security systems that leverage virtualization techniques. One of the unique features virtualization offers is the ability to observe a running operating system from an outside perspective, otherwise known as Virtual Machine Introspection (VMI). By further merging the capability with forensics memory-analysis techniques (FMA) [1], VMI is rapidly becoming a cornerstone of security for virtualized systems.

A critical problem that VMI-based security applications face is relying on data-sources that may have been tampered with. The problem is well known when using FMA techniques which often rely on information contained within the malicious guest's memory, thus also in direct reach of malware. As has been shown over the years, the OS under VMI monitoring can be subverted to break the assumptions made about the internal behavior [2], [3], thus breaking the VMI tool's capability to accurately understand the state within the guest. These types of problems have been cumulatively named the *strong semantic gap problem* [4].

With the aid of hardware based virtualization extensions it is however possible to overcome some of the limitations that FMA systems face. By establishing in-band event delivery for active observation of the guest's execution, it is possible to observe binding events, such as process scheduling [5] and kernel heap allocations [6]. However, such in-band delivery requires proper hypervisor support, which has thus been absent from the world of ARM virtualization. In this paper, we explore the hardware properties of the ARM architecture and the internal architecture of Xen, for which we have implemented such an event delivery system.

The outline of this paper is as follows. In Section II we briefly discuss relevant related work. In Section III we outline the ARM MMU virtualization extension and discuss the requirements to leverage this feature for active VMI. Continuing in Section IV we highlight critical limitations of this subsystem when used for execution monitoring. Afterwards we turn our attention to planned future work in Section V, and finally we provide some concluding remarks in Section VI.

II. Related work

While on today's CPUs, both x86 and ARM, there are a variety of options to establish in-band delivery based on hardware events, it has been a challenge over the years to find the combination of events which correspond to high-level system behavior. For example, process scheduling events are directly trappable by the hypervisor but system-calls aren't [7], [8]. The problem is largely due to the fact that the trappable hardware events often do not directly correspond to the events one actually wants to intercept, thus alternative settings need to be configured to work around the hardware limitations. However, this often leads to side-effects in the form of overhead.

For example, Ether [7] proposed a set of mechanisms that relied on changing paging permissions found in the *shadow page-tables* to trigger violations which then can be trapped to the VMM. Ether uses this mechanism to direct system calls to pages that generate trappable pagefaults by changing the `SYSENTER_EIP_MSR` register within the guest. However, due to the granularity of page permissions, this often leads to violations being triggered by events that were not the target, thus adding overhead and potential pitfalls in case these events need to be further filtered. Furthermore, shadow page table based monitoring was only able to trace read/write memory accesses performed by the guest.

Nitro [8] is another system that uses virtualization extensions to monitor system calls within the VM. Nitro, similarly to Ether, also manipulates the guest state, but instead of the `SYSENTER_EIP_MSR`, Nitro changes the `MSR_SYSENTER_CS` used during SYSENTER to cause an invalid value of 0 to be loaded into CS register. This in effect causes a general protection fault leading to a `VMEXIT`. Similar techniques were implemented for both interrupt-based and for `SYSCALL` based system-calls. The benefit of Nitro's approach over Ether is that the faults it triggers are significantly fewer then Ether's page faults, leading to better performance. On the other hand, Ether's page permission based monitoring is more flexible as it could also be used to trap other high-level events, not just system calls.

CXPInspector [9] builds exclusively on the use of hardware accelerated two-stage paging for execution tracing. Similar to shadow-page tables, hardware accelerated two-stage paging allows only for page-level granularity, however, it is now directly supported by hardware, such as Intel's Extended Page Tables (EPT) extension. As two-stage paging adds an

additional layer of paging where translation between guest physical and machine physical addresses takes place, any access violation within this layer will trigger a `VMEXIT`, thus remaining transparent to the guest. Furthermore, this new paging layer supports setting page permissions as a R/W or execute, thus it provides more flexibility then shadow page tables did. By marking certain pages non-executable (but readable and writable), CXPinspector is able to monitor the execution of the virtual machine (VM) with reduced overhead. However, the use of two-stage paging for execution tracing has been known to still add considerable overhead to the execution of the VM, mainly as a result of the granularity that can be set to trigger violations on.

SPIDER [10] evaluated the use of injecting software breakpoints to enable stealthy debugging of processes within virtual machines using the KVM hypervisor. SPIDER achieved significantly better performance than systems relying on two-stage paging for execution tracing, mainly as a result of reducing the scope of trapping. Similar to SPIDER, SPROBES [11] identified the Secure Monitor Call (SMC) instruction as one possible trapping mechanism for ARM. Unlike the exception caused by the `INT3` instruction on x86, SMC is limited to trapping code-execution in supervisor mode (that is, the execution of the guest kernel). Despite such a limitation, it has been proposed as a sufficient method for real-time kernel protection from the ARM TrustZone by researchers at Samsung [12].

In our prior work titled "Multi-tiered Security Architecture for ARM via the Security and Virtualization Extensions" [13] we outlined our plans and system design for leveraging the features discussed in the above mentioned related works. The work presented herein details the technical advancements and discoveries made in the effort to realize said architecture.

III. VMI with Xen on ARM

The three main components of VMI, as outlined by Garfinkel et al. [14], are *Isolation*, *Interpretation* and *Interposition*. Depending on the security application, *Isolation* and *Interpretation* may already be sufficient. Thus, as first step in our research effort we evaluated and ported existing applications to achieve these two components. *Isolation* on Xen is provided by the hypervisor natively, whereas security tools can operate from the privileged domain *dom0*. However, in many cases the security application - which may interact with malicious data - is also required to be isolated from the rest of the TCB. To achieve this, Xen provides a method to achieve fine-grained disaggregation of the TCB via the Xen Security Modules framework [15], with which we are able to create domains with elevated privileges, but without access to critical system resources and the hypervisor. XSM is available for ARM based devices as well, thus achieving effective *Isolation* has been easily accomplished.

For *Interpretation* our tool of choice has been the open-source LibVMI library [16], which has been specifically designed to interface with Xen and provide easy access to the guest state through the hypervisor. LibVMI accomplishes this today by utilizing a three-fold abstraction architecture: the

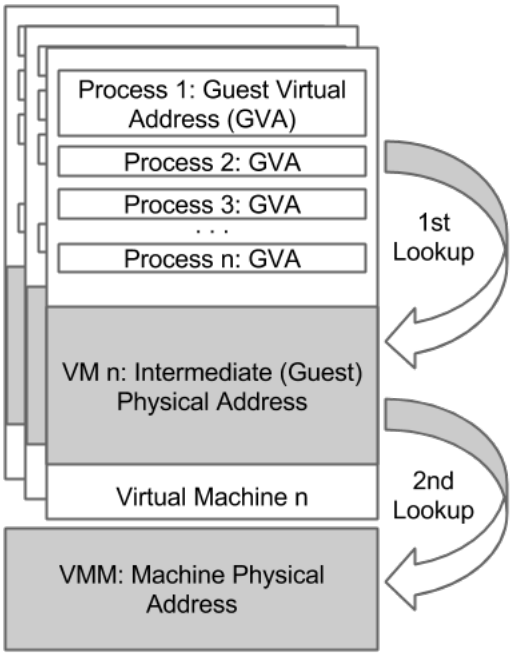

Fig. 1: 2-stage address translation

driver, the *architecture* and the *OS* layer. Effectively, the driver interacts with the hypervisor's API, the architecture layer provides guest-virtual to guest-physical address translation, and the OS layer provides reconstruction routines for standard OS features. In our scenario, LibVMI lacked the required ARM pagetable interpretation routines in the *architecture* layer. In fact, the *architecture* layer itself was not clearly defined, thus our efforts were focused on providing this extra abstraction layer with the extended support for the ARM architecture, which has been successfully ported. No modifications were required for the other components of LibVMI and everything else just worked as expected.

For effective *Interposition*, the third major component of VMI, the security system needs to configure the hardware such that it can transfer control to the security system at events of interest. In the following, we discuss the requirements for achieving such event delivery for two-stage paging based event observation with one of today's most popular open source hypervisor, Xen [17]. As Xen is a bare-metal hypervisor, the security system most likely runs in a privileged domain, but outside the hypervisor. Thus, the hypervisor needs to be extended to support the following features:

1) Setting and removing traps.
2) Customized trap handlers that understand native and artificial traps.
3) Event delivery & event notification system.

Our main focus has been enabled monitoring via the two-stage paging mechanisms, which is part of the standard ARM virtualization extension. As previously discussed, two-stage

paging allows custom permissions to be set in either stage on the page-table entry (PTE) level. Violations in the first-stage translation cause exceptions which can trap either to the guest OS or the VMM, while exceptions on the second-stage always trap into the VMM. An overview of this mechanism is shown on Figure 1. The translation itself in both stages is performed by the hardware, based on control registers. For example, on x86 the guest `CR3` register establishes which translation table should be used for stage-1, while on ARM the `TTBR1` register is used for this purpose. Second stage translation is performed similarly based on the value resulting from the 1-stage translation, only this time the page-tables reside in the VMM's memory. The ARM PTE's further allow setting permissions on R/W accesses and also provide an NX bit (eXecute Never).

Thus, to achieve monitoring of various memory accesses, such as data- or instruction-fetch accesses, the hypervisor has to expose a method to set custom permissions in the 2nd stage translation for each PTE. Xen does provide such feature for fully virtualized x86 machines (HVM) with Intel hardware supporting Extended Page Tables (EPT). Xen's internal system which allows for setting these custom permissions is dubbed *mem access*, which can optionally be enabled to deliver events on page-faults encountered during the 2nd stage translations via the *vm event* subsystem. The *vm event* subsystem uses a shared-memory ring-buffer that can be subscribed to by an observer application residing in a privileged domain. While these systems have been pre-present in Xen, in order to utilize them in the ARM architecture, both *mem access* and *vm event* had to be brought several layers up into the Xen common code, making it available for both x86 and ARM builds as well.

Significant differences have been encountered during the architecture-specific implementation for setting PTE permissions. On Intel EPT the custom permissions are stored in the EPT PTE directly in bits 58-61. This is possible because the EPT PTE entries have bits 52-62 available as software programmable bits. When an access fault is trapped, Xen can verify that the fault was triggered by the artificial traps, or by an unrelated violation by checking if the fault matches what is stored in the software programmable bits at 58-61. If it does, it is verified that the trap is an artificial trap that needs to be forwarded via *vm event*, otherwise the fault is injected back into the guest. The ARM PTEs however have a lower number of software programmable bits available, which are already occupied by other Xen features, thus an alternative permission-storage feature had to be implemented. The choice of storage was a Radix-tree based binary-lookup tree, which optimizes storage of the permissions while also providing rapid look-up on violations, with the guest physical page frame number used as the lookup key.

IV. Split-TLB on ARM

As virtual-to-physical address translation requires multiple memory accesses to walk the pagetables, and thus it's computationally expensive, modern CPUs maintain a cache for the translation results called the Translation Lookaside Buffer (TLB). The TLB stores the translation results so that subsequent lookups are sped up without having to walk the pagetables. To further improve performance, modern CPUs implement a version of the Harvard-style architecture, in which a split-TLB separates the cache into two disjoint sets: the *iTLB* stores translations for instruction fetches and the *dTLB* stores translations for data fetches.

The split-TLB architecture over the years has been used both for defensive and offensive purposes. The first system to make use of the split TLB was GRSecurity's `PAGEEXEC` [18] feature in which they tackled the problem of marking a page non-executable without explicit hardware support, like the NX-bit in later CPUs. In recent years it has also been proposed and used in similar fashion to ensure the integrity of code which reside on pages that mix code and data segments [19], [20].

In prior-work the nature of the split-TLB on x86 has been also proposed as a potential attack vector on VMI applications, in case TLB tagging is implemented [2]. The hypothesis was that in case the TLB entries remain intact after a VMEXIT/VMENTRY, a guest could maintain disjoint translation results in the iTLB and the dTLB. As an external observer has no access to the cached results directly, the VMI application would have to rely on performing translations based on the page-tables, which may no longer correspond the cached translation. However, with the recent addition of a secondary victim-cache on Intel CPUs, the *sTLB*, split-TLB based attacks are unreliable on this architecture [21], [22]. The main reason why the sTLB makes such attacks untenable, is because evicted iTLB/dTLB entries both land in the combined sTLB cache, and in case a subsequent lookup is performed, the entries from the sTLB are brought back automatically to the iTLB/dTLB. In case both the iTLB and dTLB entries are evicted into the sTLB, only one version is retained, which is what will be brought back to the iTLB/dTLB, effectively synchronizing the previously split-TLB without triggered faults that may allow the malicious guest to perform a re-split. On the ARM architecture TLB entries are tagged with the VM's ID and therefore survive VMEXIT/VMENTRY operations; however, there is no *sTLB* present, thus making split-TLB attacks a potential attack vector against VMI applications.

Further problems can be foreseen in the nature of how the ARM architecture reports access violations in the two-stage paging mechanism. While on x86 the violating guest physical address (GPA) is always reported, with the additional guest virtual address (GVA) in case paging is enabled in the guest, on ARM in most cases *only* the GVA is reported for violations. GPA is only reported on ARM when the access fault was triggered while the CPU was walking a stage-1 pagetable (aka. stage-2 fault during stage-1 translation). However, as you recall, our Radix-tree based *mem access* permission store is keyed with the guest physical page frame numbers, that can only be derived from the GPA. In order to translate the violating GVA to a GPA to look up whether the violation was triggered artificially or natively, the guest page tables have to be walked again.

On ARM there are instructions available to perform only stage-1 translation of a GVA to GPA, thus providing hardware assisted look-ups for these operations. In contrast, there is no such x86 functionality, there translations have to be done

by software by walking the guest page tables. Furthermore, performing GVA to GPA translation using hardware assisted functions on ARM also utilizes the TLB, thus under normal circumstances primed split-TLB entries should not affect the translation. That is, even if the guest page-tables no longer represent the translation that is cached in the TLB, using the hardware functions guarantees that the second lookup will result in the same translation irrespective of the modified page-tables. One problem with this ARM lookup instruction is that it performs the page-table look-up as a *data-fetch accesses*, thus only has access to the dTLB. In case of *instruction-fetch access* need to be looked up, the primed split-TLB presents various problem scenarios, as the iTLB is inaccessible outside the guest.

To illustrate the problem, let's consider a split-TLB setup where the primed dTLB entry points to a benign code-region while the iTLB points to rootkit code, with the *mem access* permission being set to alert when the rootkit code is executing. When the rootkit code is being fetched from memory, a permission fault is triggered as the stage-2 PTE disallows instruction fetches from the rootkit code. The fault handler within Xen receives the violating GVA and performs a translation via the hardware mechanism to obtain the GPA. The lookup will hit the primed dTLB entry and result in a different translation than what the actual violation-causing access cached in the iTLB is. As Xen is unable to determine if the TLB is split, the resulting translation may indicate that the code execution triggered a native exception which results in the fault being injected into the guest, forgoing notifications being sent via *vm event*. In such a case, flushing the TLB before the translation can force the translation to be performed based on the page-tables, thus avoiding hitting the primed dTLB entry. Unfortunately, when the iTLB is primed such that the page-tables no longer represent the cached translation, flushing the TLB is ineffective and the translation cannot be reproduced. This can further aid the in-guest attacker in discovering the presence of an external monitor, by observing which code executions trigger an unexpected fault. Thus, two-stage paging based execution (instruction-fetch) monitoring cannot be considered stealthy or reliable at this time.

Incidentally, while evaluating the Xen assembly routines which perform the guest virtual to guest physical address translation via the accelerated hardware functions we have discovered a critical security vulnerability that would have allowed guest systems running on 64-bit ARM systems to potentially perform privilege escalation attacks. The vulnerability has been assigned the advisory numbers `CVE-2014-3969`/`XSA-98v5` [23] and has been promptly fixed by the Xen maintainers.

V. Future work

Memory access events are only a subset of monitoring techniques available on modern hardware and the ARM platform has not yet been sufficiently explored to identify all possibilities. One already identified candidate for execution tracing is the injection of SMCs, which can be configured to trap to the hypervisor. On Xen this instruction is already configured to trap (by enabling bit 19 on the `HCR_EL2` hypervisor configuration register), thus extending support to deliver such events via *vm event* should be a relatively simple addition.

Another critical limitation in memory access based tracing is the necessity to allow the violation to progress. On x86 there are two main options for achieving that: single-stepping and emulation. In the case of single-stepping, the custom permission setting on the 2nd stage PTE would be relaxed and the hypervisor would enable the Monitor Trap Flag (MTF) to allow one instruction to progress. However, this method is only viable for single-vCPU systems, otherwise there is a race-condition between the active vCPUs. A possible solution for that would be to pause all other vCPUs while a violating one is single-stepped. However, this solution results in significant overhead. Emulation on the other hand doesn't require relaxing the custom permissions, thus avoids the need of synchronization between vCPUs.

Unfortunately, with Xen on ARM neither options are available at this time. A potential solution to this problem is the introduction of the *altp2m* subsystem, currently under development for Xen jointly by us and Intel [24]. The *altp2m* system allows defining multiple versions of the page-tables used in the 2nd stage translation, thus allowing the tables to remain intact - rather the vCPU's "view" is switched from one table to another. In such a setup, synchronization is not needed between vCPUs thus the race-condition is averted. Nevertheless, more research is required to properly identify how the execution would be reverted to the restricted table after a violation has progressed, otherwise relevant events may go unnoticed.

VI. Conclusion

In this paper we presented an overview of ARM's MMU virtualization features and how to leverage this subsystem for Virtual Machine Introspection with Xen. By developing the feature for the upcoming Xen 4.6 release we have identified several hardware limitations which can adversely affect the reliability and stealth of external monitoring applications when used improperly. We further discussed planned future work which may overcome some of the issues herein identified.

Acknowledgments

This work was supported by the Bavarian State Ministry of Education, Science and the Arts as part of the FORSEC research association.

References

[1] B. Dolan-Gavitt, B. Payne, and W. Lee, "Leveraging forensic tools for virtual machine introspection." http://www.bryanpayne.org/research/papers/GT-CS-11-05.pdf, Georgia Institute of Technology, GT-CS-11-05, 2011.

[2] S. Bahram, X. Jiang, Z. Wang, M. Grace, J. Li, D. Srinivasan, J. Rhee, and D. Xu, "Dksm: Subverting virtual machine introspection for fun and profit," in *Reliable Distributed Systems, 2010 29th IEEE Symposium on*. IEEE, 2010.

[3] T. Haruyama and H. Suzuki, "One-byte modifications for breaking memory forensic analysis," *Black Hat Europe*, 2012.

[4] B. Jain, M. B. Baig, D. Zhang, D. E. Porter, and R. Sion, "Sok: Introspections on trust and the semantic gap," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, ser. SP '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 605–620. [Online]. Available: http://dx.doi.org/10.1109/SP.2014.45

[5] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Antfarm: Tracking processes in a virtual machine environment." in *USENIX Annual Technical Conference, General Track*, 2006.

[6] T. K. Lengyel, S. Maresca, B. D. Payne, G. D. Webster, S. Vogl, and A. Kiayias, "Scalability, fidelity and stealth in the drakvuf dynamic malware analysis system," in *Proceedings of the 30th Annual Computer Security Applications Conference*.

[7] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: malware analysis via hardware virtualization extensions," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008.

[8] J. Pfoh, C. Schneider, and C. Eckert, "Nitro: Hardware-based system call tracing for virtual machines," in *Advances in Information and Computer Security*. Springer, 2011.

[9] C. Willems, R. Hund, and T. Holz, "Cxpinspector: Hypervisor-based, hardware-assisted system monitoring," Ruhr-Universitat Bochum, Tech. Rep., 2013.

[10] Z. Deng, X. Zhang, and D. Xu, "Spider: Stealthy binary program instrumentation and debugging via hardware virtualization," in *Proceedings of the 29th Annual Computer Security Applications Conference*, ser. ACSAC '13. New York, NY, USA: ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2523649.2523675

[11] X. Ge, H. Vijayakumar, and T. Jaeger, "Sprobes: Enforcing kernel code integrity on the trustzone architecture," 2014.

[12] A. M. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, and W. Shen, "Hypervision across worlds: real-time kernel protection from the arm trustzone secure world," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 90–102.

[13] T. K. Lengyel, T. Kittel, J. Pfoh, and C. Eckert, "Multi-tiered security architecture for arm via the virtualization and security extensions," in *Database and Expert Systems Applications (DEXA), 2014 25th International Workshop on*. IEEE, 2014, pp. 308–312.

[14] T. Garfinkel, M. Rosenblum *et al.*, "A virtual machine introspection based architecture for intrusion detection." in *NDSS*, 2003.

[15] G. Coker, "Xen security modules (xsm)," *Xen Summit*, 2006.

[16] LibVMI, https://code.google.com/p/vmitools, November 4 2013.

[17] Xen, http://www.xenproject.org, May 5 2015.

[18] GRSecurity, "Pageexec," https://pax.grsecurity.net/docs/pageexec.txt, December 30 2006.

[19] R. Riley, X. Jiang, and D. Xu, "An architectural approach to preventing code injection attacks," *Dependable and Secure Computing, IEEE Transactions on*, vol. 7, no. 4, pp. 351–365, 2010.

[20] J. Torrey, "More: measurement of running executables," in *Proceedings of the 9th Annual Cyber and Information Security Research Conference*. ACM, 2014, pp. 117–120.

[21] ——, "More shadow walker: Tlb-splitting on modern x86." BlackHat, 2014.

[22] T. Lengyel, T. Kittel, G. Webster, J. Torrey, and C. Eckert, "Pitfalls of virtual machine introspection on modern hardware," in *1st Workshop on Malware Memory Forensics (MMF)*, Dec. 2014. [Online]. Available: https://www.sec.in.tum.de/assets/Uploads/pitfalls-virtual-machine.pdf

[23] X. security team, "Xen security advisory 98 (cve-2014-3969) - insufficient permissions checks accessing guest memory on arm," http://seclists.org/oss-sec/2015/q1/831, March 13 2015.

[24] E. White, "Alternate p2m: support multiple copies of host p2m," http://lists.xenproject.org/archives/html/xen-devel/2015-01/msg00808.html, January 9 2015.