Virtual Machine Introspection with Xen on ARM

Tamas K Lengyel
Technische Universität München
tklengyel@sec.in.tum.de

***Abstract*—In the recent years, virtual machine introspection (VMI) 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 VMI tools. Porting these applications requires proper hypervisor support, which we have been implementing for the upcoming release of the Xen hypervisor.**

I. VMI

Virtual Machine Introspection (VMI) has been proven in the x86 world to be a viable method to create out-of-band security software. VMI relies on reconstructing high-level state information from low-level data. This data is provided by inspecting the virtual hardware components of virtual machines, such as the memory and the vCPU registers; commonly referred to as bridging the semantic gap. On the lowest level, effective VMI requires the out-of-band software to interpret the memory layout of the guest OS by understanding its paging system. Once the OS paging is interpreted, VMI can take advantage of more OS specific information to reconstruct the state of the guest.

The LibVMI [1] library was designed specifically for this purpose, supporting both Xen and KVM. Incidentally, both Xen and KVM are now available for the ARM architecture. Thus, in order to support reconstructing OS internals of ARM guests, LibVMI had been extended to support the ARM paging system.

II. Memory events with Xen on ARM

While passive observation of the target VM's memory is sometimes sufficient for some security applications (such as IDS systems); it is *interposition* that more advanced protection mechanisms rely on. For effective interposition, the security system needs to configure the hardware in order to transfer control to the security system at certain events which have been deemed critical.

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.

While Xen does provide such features for fully virtualized x86 machines (HVM), the system had to be extended to support the ARM architecture as well.

On x86, one possible method to monitor the execution of virtual machines is via *memory events*. That is, trapping the accesses made by the virtual machine during memory read/write/instruction-fetch operations. This is achieved through the two-stage paging mechanism: either with the shadow page-tables [2] or via hardware assisted nested pages (like Intel's EPT) [3]. By marking the page table entries in the second set of tables as non-readable/writable/executable, page table violations trap into the hypervisor.

With ARM's virtualization extensions, Xen runs natively on the hardware assisted nested pages. However, as compared to Intel's EPT, the page table entries contain fewer software programmable bits. This limitations is critical in our context, as on x86, the custom page permissions are saved directly into the page table entries. This approach enables the customized trap handler to easily determine whether a trap was triggered natively or by the customized permissions set by a listener application. As ARM lacks enough bits in the page table entries to store this informatiom, a separate permission-store had to be implemented, which saves the permissions.

The patches for Xen on ARM adding this new feature are being finalized and are expected to be available within Xen 4.6.

III. Future work

Memory events are only a subset of monitoring techniques available on modern hardwares, and the ARM platform has not been sufficiently explored yet to identify all possibilities. While prior art such as SPROBES [4] identified the Secure Monitor Call (SMC) instruction as one possible trapping mechanism, its utility is limited to trapping code-execution in supervisor mode (that is, the execution of the guest kernel).

Acknowledgments

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

References

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

[2] 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.

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

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