A Universal Semantic Bridge for Virtual Machine Introspection

Christian Schneider, Jonas Pfoh, and Claudia Eckert

Technische Universität München
Munich, Germany
{schneidc,pfoh,eckertc}@in.tum.de

Abstract. All systems that utilize virtual machine introspection (VMI) need to overcome the disconnect between the low-level state that the hypervisor sees and its semantics within the guest. This problem has become well-known as the *semantic gap*. In this work, we introduce our tool, *InSight*, that establishes a semantic connection between the guest and the hypervisor independent of the application at hand. InSight goes above and beyond previous approaches in that it strives to expose *all* kernel objects to an application with as little human effort as possible. It features a shell interface for interactive inspection as well as a scripting engine for comfortable and safe development of new VMI-based methods. Due to this flexibility, InSight supports a wide variety of VMI applications, such as intrusion detection, forensic analysis, malware analysis, and kernel debugging.

1 Introduction

Bridging the semantic gap requires knowledge about the virtual hardware architecture, the guest operating system (OS), or both. A VMI component that bridges the semantic gap by applying such knowledge is referred to as a *view generating component* (VGC) [6]. Most VGCs target exactly that portion of the VM state necessary for a particular VMI application, for example, protection of system hooks [7, 8], monitoring of critical data structures [3, 7], or comparison of data collected from the hypervisor and from within the guest [3–5]. Recently, cutting edge research has begun to focus on stand-alone view generation [2]. However, such work lacks the completeness and flexibility necessary for the wide range of possible VMI applications.

For our VMI research, we have implemented a VGC called *InSight* [1] to bridge the semantic gap on the Intel x86 and AMD64 platforms. In contrast to the aforementioned approaches, the goal of InSight is to make *all* kernel objects and their associated values available with as little human effort or expert knowledge as possible. InSight strives to interpret the entire kernel memory of a running guest OS in the same way the guest kernel does. It is very flexible in that it provides access to the VM state through a powerful command line interface as well as through a scripting engine to directly interact with kernel objects as JavaScript objects. This way, the actual VMI application is completely decoupled from the view generation process, making InSight a universal tool for various applications such as intrusion detection and forensic analysis, as well as for kernel debugging.

2 Challenges in View Generation

Reconstructing a VM's state from the vantage point of the hypervisor requires the following knowledge of the guest OS as well as the virtual hardware architecture: (1) the layout of the kernel address space, (2) the layout and size of kernel data structures, (3) the location of kernel objects in the kernel's address space, and (4) the function of the virtual-to-physical address translation. This knowledge is delivered to the VGC out-of-band—that is, before the introspection starts [6]—in the form of, for example, kernel debugging symbols and a software-based memory management unit. The view is then generated by applying this knowledge to the guest physical memory. When put into practice, re-creating the VM's state with this knowledge at hand provides several challenges, as we will outline in the following. Up until now, these challenges have been solved by manually applying expert knowledge on a case-by-case basis.

OS kernels organize objects in efficient data structures such as, for example, linked lists or hash tables. A common implementation of these lists and tables works by defining generic "node" data structures along with auxiliary functions and macros to operate on them. These nodes must then be embedded as a member of the kernel object's definition to be organized in a list or table. To retrieve objects stored in such data structures, the auxiliary functions operate only on the embedded nodes but still make the embedding object available through address manipulations and type casts. When a VGC aims to make the objects stored in such a list or table available, it faces several problems. First, the head to such data structures is often a plain node itself. There is no hint in the debugging symbols as to which object type is stored in a particular list. Second, once the VGC has access to an object stored in a list or table, the type information does not indicate which object type lurks behind the pointers of the embedded node member. To make matters worse, many structures contain several node members as heads to other lists holding various types, making it impossible to guess the correct type based on heuristics.

Further uncertainties in inferring the correct object type and address result from untyped pointers, pointer casts from integer values, offset pointers, pointers with status flags encoded in their least significant bits, pointers that are used as arrays, and ambiguous types such as unions.

OS kernels perform various operations atomically or in critical sections to keep their state consistent. Detecting that the virtual CPU is currently in a critical section is a difficult task, as the debugging symbols do not provide enough information to discover such situations. In case a view is generated while the CPU is in a critical section, the VGC might encounter inconsistent data structures; this is even more likely for a VM with multiple CPUs. It must then be able to cope with such inconsistencies in a reliable way.

3 Implementation

For our VMI research, we need a view-generating component that automatically provides access to *all* kernel objects in memory, not only to a small number of selected data structures. In addition, we require a high-level, flexible, and powerful interface to these objects in oder to be able to perform sophisticated analysis of the guest OS state. Our tool, InSight, is such a view-generating component for the x86 and AMD64 platform that addresses many of the aforementioned challenges. It is written in C++ and operates on physical memory. Thus, it works with memory dump files as well as with any hypervisor that provides access to its guest's physical memory. For Linux guests, InSight fully supports the reading of arbitrary kernel objects for both 64-bit and 32-bit addressing schemes with and without physical address extension (PAE). Multiple memory files can be processed simultaneously (for example, to compare the current "live" state of a running VM to a previous snapshot).

We designed InSight to be as straightforward to use as possible. Whenever the member of a kernel object is accessed, the user receives a new kernel object of the type that this member represents; all pointers are automatically dereferenced. With regard to efficient lists and hash tables that are implemented as detailed in Section 2, InSight detects these "node" data structures and marks the particular members of the embedding structure accordingly. If the user accesses these members, he receives an object of the embedding type with its correct address, just as the kernel macros for operating on those data structures would do.

The current approach enables an easy enumeration of running processes and loaded kernel modules, for example. In situations in which total automation is not achieved, the flexible design of InSight allows the user to address this by encapsulating expert knowledge in JavaScript functions that are reusable for all applications. In an effort to reduce the amount of human intervention even further, we are already working on an improved solution which parses the kernel's source code in order to automatically resolve virtually all data types of pointers and type casts without requiring expert knowledge.

The InSight shell provides an interactive interface to analyze the state of a VM. It allows inspection of the kernel's data structures and global variables. For each loaded memory file, the kernel objects referenced by the global variables as well as their members can be accessed using the common notation `object.member`. In addition, the user may choose to read an object from an arbitrary virtual or physical address as any known data type, allowing the user to further follow any members of that type.

The true power of InSight lies in its included JavaScript engine which enables users to interact with kernel objects and to automate complex analysis tasks. The user can access global variables by requesting an `Instance` object of a variable by its name. If the resulting object represents a structure, its members can be accessed by their name using the "dot" notation just like any other property of a JavaScript object. In addition, an `Instance` object provides methods to access meta data of the corresponding kernel object, change its type, manipulate its address, and access array elements. Using all of these features, writing a function that prints out a list of loaded kernel modules requires less than ten lines of JavaScript code. Besides its simple application, another substantial benefit of the scripting engine in comparison to any low-level approach is the fact that any sort of runtime error will result in a JavaScript exception rather than a segmentation fault. All errors are contained within the scripting engine and do not propagate to the VGC itself. The user can modify and re-run the script instantly without having to recompile the source code, restart InSight, or restart the monitored VM. This is a major advantage of InSight over other approaches and greatly eases the development of new VMI mechanisms.

4 Conclusion

InSight focuses on the well-known semantic gap issue faced by all VMI applications. While other approaches strive to bridge the semantic gap for a particular problem, InSight takes a general approach. We strive to provide a VGC that gives a complete view of the guest kernel, is decoupled from the VMI component itself, and remains flexible enough to be applicable for forensics, intrusion detection, malware analysis, and kernel debugging. Our efforts have resulted in InSight which is an extremely powerful and flexible VGC offering several interfaces including a scripting engine for further automation of complex tasks. It has already proven to be very useful and is successfully applied in several projects within our research group. We have released InSight as an open source tool [1] to enable the fast and intuitive development of new VMI and forensic approaches.

References

1. InSight project website, `https://code.google.com/p/insight-vmi/`
2. Dolan-Gavitt, B., Leek, T., Zhivich, M., Giffin, J., Lee, W.: Virtuoso: Narrowing the semantic gap in virtual machine introspection. In: Proceedings of the IEEE Symposium on Security and Privacy (Oakland) (May 2011)
3. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: Proc. of NDSS. pp. 191–206 (2003)
4. Litty, L., Lagar-Cavilla, H.A., Lie, D.: Hypervisor support for identifying covertly executing binaries. In: Proc. of the 17th conf. on Security symp. pp. 243–258. USENIX, Berkeley, CA, USA (2008)
5. Martignoni, L., Fattori, A., Paleari, R., Cavallaro, L.: Live and trustworthy forensic analysis of commodity production systems. In: Proc. of 13th Int. Conf. on Recent Advances in Intrusion Detection. pp. 297–316. RAID'10, Springer (2010)
6. Pfoh, J., Schneider, C., Eckert, C.: A formal model for virtual machine introspection. In: Proc. of 2nd Workshop on VM Sec. ACM, New York, NY, USA (2009)
7. Riley, R., Jiang, X., Xu, D.: Guest-transparent prevention of kernel rootkits with VMM-based memory shadowing. In: Recent Advances in Intrusion Detection, LNCS, vol. 5230, pp. 1–20. Springer (2008)
8. Wang, Z., Jiang, X., Cui, W., Ning, P.: Countering kernel rootkits with lightweight hook protection. In: Proc. of 16th Conf. on Computer and Communications Security. pp. 545–554. CCS '09, ACM (2009)