Digital Investigation 11 (2014) S85–S94

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

VMI-PL: A monitoring language for virtual platforms using virtual machine introspection

Florian Westphal [a, b, *], Stefan Axelsson [a], Christian Neuhaus [b], Andreas Polze [b]

[a] Blekinge Institute of Technology, Sweden
[b] Hasso-Plattner-Institute, University of Potsdam, Germany

Keywords:
Virtualization
Security
Monitoring language
Live forensics
Introspection
Classification

A B S T R A C T

With the growth of virtualization and cloud computing, more and more forensic investigations rely on being able to perform live forensics on a virtual machine using virtual machine introspection (VMI). Inspecting a virtual machine through its hypervisor enables investigation without risking contamination of the evidence, crashing the computer, etc. To further access to these techniques for the investigator/researcher we have developed a new VMI monitoring language. This language is based on a review of the most commonly used VMI-techniques to date, and it enables the user to monitor the virtual machine's memory, events and data streams. A prototype implementation of our monitoring system was implemented in KVM, though implementation on any hypervisor that uses the common x86 virtualization hardware assistance support should be straightforward. Our prototype outperforms the proprietary VMWare VProbes in many cases, with a maximum performance loss of 18% for a realistic test case, which we consider acceptable. Our implementation is freely available under a liberal software distribution license.

Introduction

More and more computing services are being provided remotely in the form of so called "cloud based services" in the form of *virtual* machines, as it is done by Windows Azure, or Amazon EC2, etc. Thus virtualization is becoming an evermore important tool for providing computing services to customers, especially when it comes to providing e.g. web servers, data base servers etc. so called *infrastructure as a service (IaaS)*. This market segment is growing fast, at present.

Hence, the probability that forensic investigations will have to be performed on virtual machines will continue to increase. Whereas virtual machines can be treated and analyzed as normal computers (Flores Cruz and Atkison, 2011), their use in cloud settings poses several challenges, and also opportunities for the investigation. These challenges include legal issues, such as cross-border legislation problems (Biggs and Vidalis, 2009), as well as trust issues (Dykstra and Sherman, 2012).

Despite these challenges, virtual machines also have an advantage over real computers when it comes to live forensics. Whereas live forensics on a real machine always causes state changes, possibly contaminating the evidence (Adelstein, 2006), live forensics on a virtual machine can be done with minimal interference with its state, by using virtual machine introspection (VMI) (Cruz and Atkison, 2012). Several VMI techniques have been developed and implemented, in recent years.

To this end we have developed a domain specific language (DSL) named VMI-PL, that can be used to define and apply common VMI techniques for the monitoring of virtual machines. This DSL provides a simple way for the user

* Corresponding author. Hasso-Plattner-Institute, University of Potsdam, Germany.
E-mail address: florian.westphal@student.hpi.uni-potsdam.de (F. Westphal).

http://dx.doi.org/10.1016/j.diin.2014.05.016
1742-2876/

to specify virtual machine events which should be intercepted, registers and virtual memory, which should be read or written to, and data streams from and to the virtual machine, which should be logged. In addition to data about the hardware level, the language provides means to obtain operating system level information by interpreting the data available to the hypervisor. This combination of ease of use and capabilities it provides makes it, in our opinion, a valuable tool for live forensic investigations.

In connection with the language design, we propose a new classification for VMI techniques, which is based on the respective key mechanism of the VMI technique. These key mechanisms can be data accesses, hardware events, or data transfers across the border between virtual machine and hypervisor. In addition to these theoretical contributions, we also developed a prototype, implementing our language design in the Kernel-based Virtual Machine (KVM) hypervisor, which is freely available under an open source license.

Related work

As VMI-PL allows the implementation of new monitoring strategies without modifying the hypervisor further beyond that which was necessary to implement support for VMI-PL. We limit the related work section to those approaches that share this characteristic. This excludes approaches such as HyperDbg (Fattori et al., 2010) (which is not a true hypervisor based system), and VMST (Fu and Lin, 2012) (that while their approach has much to commend it, is limited to just the one read-only strategy of memory introspection). We have only found two other publicly available solutions for which this is true: LibVMI,[1] and VMware VProbes (VMware and Inc, 2011).

LibVMI is an introspection library for Xen and KVM, which was derived from the XenAccess project (Payne et al., 2007). While Xen is fully supported by this library, the support for KVM is more limited, due to missing hooks in the KVM code. With LibVMI, it is possible to read and write virtual memory, as well as to register handlers for certain events occurring within a virtual machine. These events can be memory accesses, register writes, or instruction execution, where instruction execution can be tracked either in single stepping mode or by specifying a particular virtual address.

VProbes, on the other hand, is a monitoring solution for VMware Workstation and VMware Fusion, which provides the possibility to specify probes, which are executed on certain events. The specification of these probes is done in VMware's own language called *Emmett*. While the initial idea of VProbes was to support external debugging of virtual machines (Adams, 2007), it has also been shown to be useful in the context of e.g. intrusion detection (Dehnert, 2012). It should be noted that *Emmett* only allows the reading of information from the monitored host.

In contrast to VProbes, our monitoring language supports writing to effect the manipulation of the internal state of the virtual machine.

[1] https://code.google.com/p/vmitools/.

Our monitoring language differs from both these solutions in that it provides access to operating system level information and events. This simplifies investigations, since the interpretation of the hardware state is done automatically. Another benefit from our approach is that it monitors data streams from and to the virtual machine, which cannot be monitored using the other solutions. This feature enables an investigator to monitor network traffic, as well as the keyboard input of a user. Finally, our monitoring language has a simple syntax, which enables a user to obtain data from a virtual machine without the need to have an extensive background in low level programming.

Virtual machine introspection

Virtual machine introspection (VMI) was coined by Garfinkel and Rosenblum (Garfinkel and Rosenblum, 2003) and was defined by Pfoh et al. as a "*method of monitoring and analyzing the state of a virtual machine from the hypervisor level.*" (Pfoh et al., 2009). Since VMI obtains this state information directly from the hypervisor, the semantic knowledge of the guest operating system's abstractions is lost. Hence, it is difficult to interpret the data. This was first described by Chen and Noble as the *semantic gap* problem (Chen and Noble, 2001). In order to overcome the semantic gap, there are different strategies, which were categorized by Pfoh et al. (Pfoh et al., 2009) as *out-of-band delivery*—where the needed semantic knowledge is supplied externally, *in-band delivery*—that pass the needed data along with the semantic information directly from inside the guest operating system to the hypervisor, and *derivation*—that derives the needed information from how the guest operating system is using the given hardware interface.

These approaches have different advantages and disadvantages when it comes to the amount of information they can easily provide and how difficult it is for an attacker to deceive them. Additionally, it should be noted that *in-band delivery* techniques are less suitable for live forensics, since they require the running of software inside the virtual machine, potentially manipulating its state to a considerable extent, and therefore neglecting the advantage VMI normally provides for live forensics.

VMI techniques

For reasons that will become clear we decided to further refine this classification of VMI techniques.

While all techniques described in the following sections can be categorized as *out-of-band delivery* or *derivation* techniques, we will divide them further into *data-based*, *event-based*, and *stream-based* techniques. This categorization has been useful in the specification of our monitoring language, since it takes into account the underlying mechanisms of the particular VMI technique.

Data-based techniques

Data-based VMI techniques focus on the virtual storage of a virtual machine, such as registers, main memory, and the virtual hard disk. These techniques obtain information from the virtual machine by reading that storage and