Secure Architecture for VMI-based Dynamic Malware Analysis in the Cloud

Benjamin Taubmann, Hans P. Reiser

Secure Architecture for VMI-based Dynamic Malware Analysis in the Cloud

Benjamin Taubmann, Hans P. Reiser
University of Passau
Passau, Germany
Email: {bt,hr}@sec.uni-passau.de

I. Introduction

A popular approach for automated malware analysis, as for example used by the Cuckoo Sandbox [1], is to use agents within the target system of the malware to collect analysis data. Such agents can easily be detected and attacked by malware. Virtual machine introspection (VMI) has recently gained much attention as a more stealthy approach to malware analysis, for example in DRAKVUF [2] and DECAF [3].

Using VMI requires adequate privileges for accessing other virtual machines. As a result (almost) all VMI-based approaches for malware analysis run the analysis tool in a privileged system domain, such as Dom0 on Xen systems and the host OS on KVM systems. Modern analysis tools have a large complexity and thus might contain exploitable vulnerabilities with a non-negligible probability. Attacks that exploit analysis tools by manipulating the observed data are a real threat, as demonstrated by examples such as CVE-2006-5276[1]. This means that running such analysis tools on a privileged domain puts the whole system at risk.

We propose a different approach to VMI-based malware analysis that is more suitable for using in a large scale or even in a public cloud. Even if the analysis tools using VMI are faulty, they should not put the whole environment at risk. With adequate isolation approaches, it is even possible to deploy customer-defined VMI tools in public cloud environments.

II. Research Goals

In this paper, we sketch an outline of our secure architecture for VMI-based dynamic malware analysis, which is well-suited for running the analysis on dynamically allocated cloud resources.

First of all, we aim at obtaining an architecture in which the analysis tools are fully isolated from the core system infrastructure, i.e., instead of running such tools in privileged domains such as Dom0 on Xen systems, we want to run them in a fully isolated domain, in order to minimize the impact of a compromised analysis tool. Similarly, we want to support the simultaneous analysis of multiple virtual machines. In this case, we additionally want to ensure that one of the analysis runs cannot influence other analysis. This means that we also want isolation between multiple analysis instances.

Furthermore, a goal of our architecture is to offer adequate integration into a cloud management infrastructure. A cloud client should be able to dynamically allocate resources required for analysing malware samples running in a different virtual machine. This allocation requires assigning appropriate privileges.

Finally, our system aims at jointly analyzing a target VM's main memory, disk content and network traffic and combine these results. This combination enables new features such as deep process introspection by combining, e.g., symbol information from library and executable files from disk with process memory introspection, and decryption of network traffic using key information extracted from main memory.

III. Architecture

Figure 1 shows an overview of our architecture. From a top-level perspective, the main features of the architecture are the isolation between analysis VMs ("Monitor VMs" and "Sandbox VMs") and the automated cloud deployment. Specific analysis components focus on main memory, disk storage, and network, and combine data from these sources.

A. Isolation

In our architecture, we use dedicated monitoring virtual machines for analyzing the behaviour of a target virtual machine on which the malware sample is deployed (sandbox VM). We use the Xen security modules (XSM) extensions for controlling the access between virtual machines. XSM is based on mandatory access control, which implies that a static set of types for virtual machines and a set of rules for enabling VMI access from one monitoring VM to one target VM are defined statically.

This rule set ensures that the virtual machine that runs the analysis code has access to only a single target VM. Even if, due to some vulnerability in the analysis code, malware can compromise the monitoring VM, it cannot easily compromise other domains or the system's Dom0.

B. Deployment

A central controller instance orchestrates the analysis of (potentially many) malware samples on multiple target virtual machines. Our prototype is based on the OpenNebula cloud management system, which enables us to automatically deploy analysis on multiple hosts of a private cloud.

The controller uses the OpenNebula API for launching monitoring VM and target VM. We have developed extensions that automatically map a matching pair of XSM labels to both

[1] http://www.cvedetails.com/cve/CVE-2006-5276/ [accessed 2016-03-21]

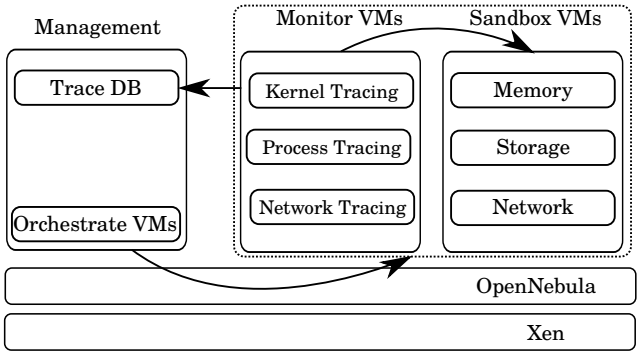

Fig. 1. Cloud-based architecture for VMI-based dynamic malware analysis. A central management controller instantiates pairs of sandbox VM and monitor VM. The monitor combines information obtained at the level of main memory, disk storage and network traffic of the sandbox VM.

VM instances such that VMI access is enabled from a monitor VM to the corresponding sandbox VM.

C. Memory

For static memory analysis of the target VM, our architecture uses well-known memory analysis tools, in particular Volatility[2]. These tools mainly focus on extracting information from the guest operating system, such as extracting lists of running processes, loaded kernel modules, or active network connections.

In addition to that, our architecture aims at providing support for dynamic analysis of the sandbox VM. For that purpose, it offers basic mechanisms for tracing the execution of the target VM by injecting software breakpoints and by tracing memory events.

On top of that, our architecture provides mechanisms for deep introspection both at the kernel level and application level. At the kernel level, we can, for example, trace the execution of selected system calls ("kernel tracing"). At the application level, we can trace specific functions from the application or from a library ("process tracing").

D. Storage

Application-level function tracing requires access to the function symbols of an application, which are stored as exported symbols in the executable or library binary on disk, but not in main memory. In addition to accessing main memory, we thus need to extract these symbols from files on disk whenever executable code is loaded from disk storage.

Furthermore, it is a very effective approach to monitor the file system access of a VM for malware analysis as some samples store their unpacked version before the execution on disk. Additionally, the information of accessed files can be used to determine if the malware decrypts other files or tries to inject itself into other applications.

[2]http://www.volatilityfoundation.org [accessed 2016-03-21]

E. Network

The network traffic is a very useful source in order to analyze the behavior of malware samples. Thus, we store the traffic for further investigations. Additionally, monitoring the network traffic allows the reconstruction of a VM state at a higher level than monitoring low level traces. For example it is cheaper to monitor the network traffic than monitor all `read` and `write` system calls. Additionally, this approach is platform independent and does not require any information how the operating system handles network connections. This approach was used by TLSkex to trigger memory snapshots when specific network packets occur [4] in order to extract TLS session keys from memory.

IV. Conclusion

We have presented an architecture that enhances dynamic malware analysis by combining data from three sources: target VM main memory, disk storage, and network traffic. This combination yields additional insights, such as traces of specific functions of selected applications and the decryption of network traffic based on key extraction.

Our architecture has security advantages compared to other state-of-the-art approaches. It uses virtual machine introspection, and thus our analysis tools are better isolated from malware than in systems that use in-target agents. It allows monitoring multiple targets in multiple VMs simultaneously, but all analysis instances use separate virtual machines and thus are strongly isolated one from each other. Unlike most other VMI-based systems, our architecture does not run VMI analysis code on a privileged domain (such as Dom0 on Xen-based systems), and thus does not cause any risk even if the analysis system is compromised.

Acknowledgment

The research leading to these results was supported by the Bavarian State Ministry of Education, Science and the Arts as part of the FORSEC research association.

References

[1] D. Oktavianto and I. Muhardianto, *Cuckoo Malware Analysis*. Packt Publishing, 2013.

[2] 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*, 2014.

[3] A. Henderson, A. Prakash, L. K. Yan, X. Hu, X. Wang, R. Zhou, and H. Yin, "Make it work, make it right, make it fast: Building a platform-neutral whole-system dynamic binary analysis platform," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ser. ISSTA 2014. New York, NY, USA: ACM, 2014, pp. 248–258.

[4] B. Taubmann, C. Frdrich, D. Dusold, and H. P. Reiser, "TLSkex: Harnessing virtual machine introspection for decrypting TLS communication," in *Proc. of the DFRWS EU 2016 Annual Conference*, 2016.