VMM detection through samepage merging

Daniel Fernandez (soyfeliz@48bits.com)

Abstract

Virtual environments are commonly used in malware analysis. Several techniques have been developed to detect and evade them in order to make malware harder to analyze. Some VMMs take advantage of hardware virtualization making them harder for malware to detect them. This paper introduces a new method to detect VMMs including those based on hardware virtualization.

Introduction

Behavioral differences exist between virtualized environments and real hardware. These differences form the techniques used in VMM detection. VMMs using hardware virtualization capabilities are harder to be detected because fewer differences are exposed.

Some methods used to detect hardware virtualization based VMMs use timing analysis over resources like caches and memory [1]. For example, measurements can be made on memory access times with CPU cache enabled and "disabled". Normally the VMM won't allow the cache to be disabled, so if the results of both measures are similar it means we are inside a VM guest. The method proposed here is similar, but is far simpler and can be done from ring3; however, it has some limitations. Only few VMMs can be detected using this method (tests were made only on VMware [2] and KVM [3]).

Same content sharing

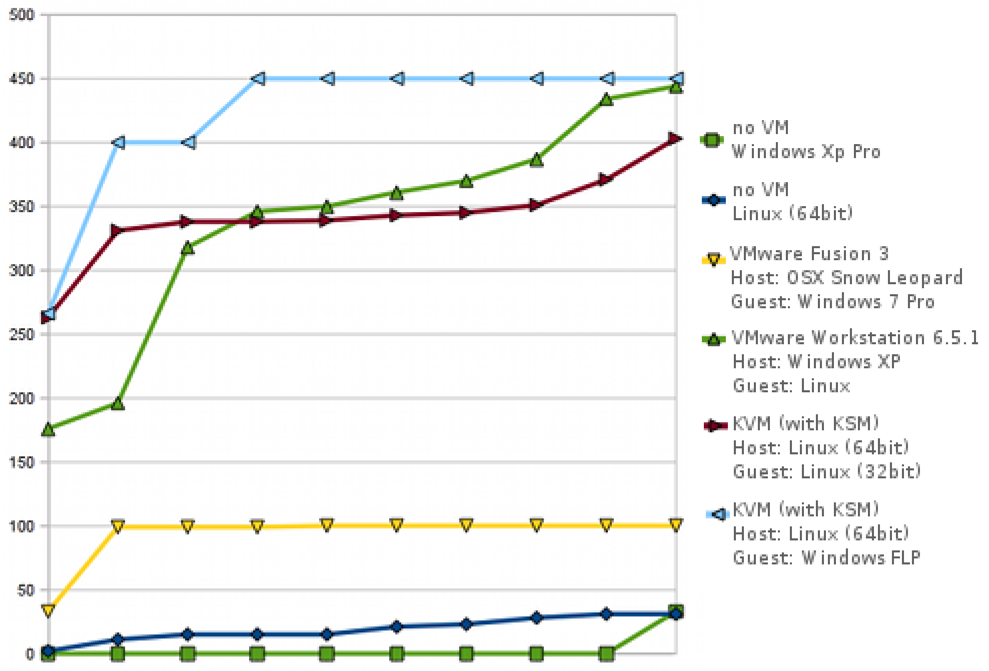
In order to save memory, the OS provides a system to share page frames known as COW (Copy on Write). The same physical memory is used across different tasks, the pages are marked read-only and when a task tries a write, the exception handler of the OS assigns a new page for that task.

The VMM periodically scans guests' memory and COW-shares all pages with same content. The computational cost of this operation is too high to be used in OS memory management, but is an attractive option for virtualization, mostly when running multiple guests. The original idea seems to belong to VMware [4]. Later, Linux introduced an implementation called KSM [5] (Kernel Samepage Merging). It can be enabled and used in any recent version of KVM (and maybe by other VMMs in the future).

Detecting differences in memory access times

A big buffer can be prepared filling all pages with the same pattern and then a measurement of the time needed to write the whole buffer is taken. The above procedure is repeated, this time, waiting while the VMM merges the pages before the write operation. Then the difference between both measurements is calculated.

The following graph show the results of ten tests run in different environments:



According to the results, there is a considerable time difference when running the tests inside and outside the VM. This is due penalties when breaking the COW on the written pages. There are differences between results of the same tests too, due to other factors like system load. An interesting case occurs with OSX, where the difference is just 100%, being close to the results outside the VM. Even in a test it reached 33%, making it a very difficult target to be detected.

Looking at the graphs and not considering OSX results, a threshold of 150% could be used to detect the VMM. Taking OSX results in consideration, I decided to choose a threshold of the 98%. Even so there may be false negatives on OSX and such a low threshold could cause false positives if certain conditions are met.

Conclusion

This method could be used in order to detect some kinds of VMMs, while is not 100% effective, it can be refined more. Heuristics could be used to adjust the threshold and other parameters, based on extra information like the system load. One problem is the amount of time needed before making the second measurement, but for malware authors that would not be a limitation.

Resources

The program used to run the tests can be found here: http://www.48bits.com/projects/smdetect.cpp

It can be compiled on Linux and Windows (minGW).

References

1. Thomas Raffetseder, Christopher Kruegel, and Engin Kirda: Detecting system emulators. (2007)
2. Vmware Inc. http://www.vmware.com/ (2009)
3. KVM: Kernel Based Virtual Machine. http://www.linux-kvm.org/ (2009)
4. Carl A. Waldspurger: Content-based, transparent sharing of memory units. (2004)
5. KSM - Linux Kernel Documentation. http://www.kernel.org/doc/Documentation/vm/ksm.txt (2009)