

Patagonix: Dynamically Neutralizing Malware with a Hypervisor

H. Andrés Lagar-Cavilla - andreslc@cs.toronto.edu
Lionel Litty - llitty@cs.toronto.edu

Introduction

In most operating systems widely in use today, it is possible for malicious software (malware) that has gained administrator privileges on the system to arbitrarily modify the kernel. Recently, Windows-based malware has been increasingly able to wedge itself deeper into the system by taking advantage of this failure of the operating system (OS) to protect itself. This results in the malware being both increasingly stealthy and hard to uninstall, because code running in user space cannot rely on the kernel's integrity anymore. In the extreme, pernicious malware could make it close to impossible to repair the system, and a full reinstall of the system may be necessary.

In this project, we plan to explore the feasibility of neutralizing malware by leveraging a hypervisor, a thin layer of software running below the operating system. Ideally, by leveraging knowledge of the architected constraints of the platform, a hypervisor might be able to neutralize malware using only minimal knowledge of the inner workings of the monitored OS. Our project will focus on a hypervisor that relies on hardware support (Intel Virtualization Technology VT-x) to run unmodified host operating systems. Such an approach would then enable one to dynamically clean systems of malware that has taken steps to hide itself/prevent its removal from user space without even requiring a reboot of the system.

Approach

Our approach will rely on manipulating the permission bits in the architected page tables that an operating system must use when using virtual memory on the x86 platform. By modifying these bits in the shadow page tables maintained by the hypervisor, a hypervisor patch (dubbed Patagonix) can efficiently inspect all binary code run on the system and keep track of exactly what code is being run on the system. In addition, Patagonix can dynamically inject code in processes running on the system to modify their behavior.

We envision two usage scenarios: in one scenario, the user of Patagonix wants to allow only some applications to run and has a list of hashes of code pages corresponding to these applications. For example, the user wants to run a malware scanner inside a monitored VM and wants to make sure that only the malware scanner, the operating system and maybe crucial daemons can run while scanning the system. In the second scenario, the user knows the hashes that correspond to a piece of malware she wants to neutralize. Patagonix either makes sure that only blessed code can run or that the malicious code cannot run. To prevent code from running, Patagonix can either return illegal instruction interrupts to the OS when suspicious code tries to run, or attempt to neutralize that code by injecting instructions in the faulting page and changing the current instruction pointer (EIP) to point to that injected code.

In addition to these usage scenarios, Patagonix can provide a user with useful information, such as exactly which code pages it saw during a time interval or which code pages it has seen since boot.

Project roadmap

We will use Xen as our hypervisor. Xen can run unmodified operating systems such as Windows, as long as hardware support for virtualization is available. We will start by studying the shadow page table implementation in Xen and insert hooks in that implementation to allow Patagonix to manage permission bits in the monitored OS's page tables. We will also study Windows' Portable Executable (PE) file format to be able to easily extract the hashes of memory pages from a binary on disk. We will finally study the feasibility of suspending undesirable applications by having Patagonix inject code. The injected code will be Windows specific.

If we get all these pieces working, we will then be able to test our approach by attempting to suspend malware known to be hard to remove from user space, such as Look2Me. We will also attempt to

suspend all running applications in a system except for a malware scanner.

Bibliography

Lionel Litty and David Lie. It's 2006: Do You Know What Your Computer is Doing? In **ASID** 2006 Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In Proceedings of the 13th USENIX Security Symposium, August 2004.

Stephen T. Jones, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Antfarm: Tracking processes in a virtual machine environment. In Proceedings of the 2006 Annual Usenix Technical Conference, June 2006.

Stephen T. Jones, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Geiger: Monitoring the page cache in a virtual machine Environment. In ASPLOS 2006