Descriere pentru licenta

Bodea Nicolae, grupa 922

Protection based on hardware exceptions at hypervisor level

Introduction

The hypervisor level protection is a very reliable technique mainly because of the absolute isolation of the guest code (in this case Windows) from the hypervisor code. The hypervisor level is usually called the “ring -1”, but in reality it is just code executed on ring 0 that uses the VMX capabilities of Intel CPU when executing code from guest. Thus, the hypervisor code is called VM Root Operation and the guest code is called VM Non-Root Operation. Firstly, the hypervisor will start the VM Root Operation and define some conditions on which VM Non-Root Operation will cause a VM Exit, this meaning that the guest code will trigger the mechanism which makes the processor jump to hypervisor code. This is very effective because good conditions for VM Exit may be defined such that in cause of a security concern, the hypervisor can verify if the operation made by the guest was malicious, thus protecting the guest from attacks that a normal anti-virus will fail to protect.

In theory, it sounds perfect, but the main problems of this level of protection are performance and guest communication. Performance can be obtained by staying as less as possible in VM Root Operation and to make less VM Exits, which may be a problem because less VM Exits means that there are less conditions that are verified, thus a lower level of security is obtained. This is a parallel to the “time-memory complexity reducing” problem, where reducing complexity of time can cause increasing memory complexity and vice-versa. Making a hypervisor more performing can cause security problems and making a hypervisor attain full security can be a performance killer.

Guest communication can be obtained by having a driver in kernel mode which communicates with the hypervisor based on some predefined protocol. A good way to ensure communication is by using VMCALL instruction (which causes an exit), and put a magic for validation in a CPU register, and a command in another register. For this purpose, on this project I have used RCX as the command register.

Security techniques

The security techniques used on this project will be the explicit treating of hardware interrupts at hypervisor level. We can configure the VMCS such that for every hardware exception a VM Exit occurs. That means that every time a hardware exception occurs we will execute a privileged routine in the hypervisor code which would handle the exception, and depending on what the outcome of the exception handle being executed would be, we will block the action or let the action happen. A good example for this would be the Page Fault

exception. A page fault exception (#PF) occurs when an instruction is executed on non-mapped memory. This is usually the case of a page swap, so the windows interrupt service routine for page fault will search on swap file the page which the instruction needs and fetch it into memory. This is a good mechanism, but an attacker may intervene in this process, for example having a process which is aware of other processes addresses in memory, it can provoke a page fault which swaps a page entirely into another process memory. Imagine that you have a process “attacker.exe” which issues a page fault on the page “0x7ff2f3000” which belongs to the process “chrome.exe”. If the instruction pointer for chrome points on this page, it will execute the code which “attacker.exe” desires, but in the context of “chrome.exe”, so for example an attacker can cause more malware to be downloaded and the culprit would be chrome, but this could be avoided by verifying who issued a page fault for which page and block actions like swapping a page from a process to another process. Using this technique we can also be aware of, and block hooks put in user mode by issuing page faults on write-on-copy actions. This is very extendible as more attack vectors can be used, basically for each hardware exception an attack can occur.

What will the project contain?

Firstly, the main component of the project will be the hypervisor which will start the guest and ensures the security. Secondly, a Windows driver will be used for hypervisor communication, such that the guest will be aware of the attacks found by the hypervisor. Finally a graphical interface will be used as an admin panel for the user. This panel communicates with the driver via a network port and will let the user know about the attackers found, and will let the user decide for each attack what to do (the attack will be blocked before, but the user can decide to delete the attacker). By having the driver in kernel mode, thus privileged, when the user decides to eliminate the attacker, the driver can terminate the problem process and delete the executable file of the attacker without requiring any more permissions from the operating system.