HyBIS: Advanced Introspection for Effective Windows Guest Protection

Roberto Di Pietro, Federico Franzoni, Flavio Lombardi

HyBIS: Advanced Introspection for Effective Windows Guest Protection

Roberto Di Pietro[1,2] Federico Franzoni[3] and Flavio Lombardi[4]

[1] Nokia Bell Labs, Paris-Saclay, France
[2] Università di Padova, Maths Dept., Padova, Italy
[3] Pompeu Fabra University, Barcelona, Spain
[4] IAC-CNR, Rome, Italy

Abstract. Effectively protecting the Windows^{TM} OS is a challenging task, since most implementation details are not publicly known. Windows OS has always been the main target of malware that have exploited numerous bugs and vulnerabilities exposed by its implementations. Recent trusted boot and additional integrity checks have rendered the Windows OS less vulnerable to kernel-level rootkits. Nevertheless, guest Windows Virtual Machines are becoming an increasingly interesting attack target. In this work we introduce and analyze a novel *Hypervisor-Based Introspection System* (HyBIS) we developed for protecting Windows OSes from malware and rootkits. The HyBIS architecture is motivated and detailed, while targeted experimental results show its effectiveness. Comparison with related work highlights main HyBIS advantages such as: effective semantic introspection, support for 64-bit architectures and for recent Windows versions ($\geq$ win 7), and advanced malware disabling capabilities. We believe the research effort reported here will pave the way to further advances in the security of Windows^{TM} OSes.

1 Introduction

Securing the Windows OS is a very challenging task, given its complexity and also given that its internals are not publicly known. Over time, a large set of malware has targeted vulnerabilities in Windows OSes and services. Due to the very large installed base of Windows OSes, there is a great amount of new malware produced every year, which implements advanced methods for detection avoidance. The problem is particularly interesting for recent Windows versions, which have not yet been fully analyzed/investigated by the research community.

Among the different kinds of malware, rootkits represent the most complex and dangerous threats. In fact, rootkits can alter the system's perception of itself, and conceal malicious activities over a large period of time. In particular, modern rootkits can directly manipulate memory structures to further enhance their stealthiness. As such, security tools can hardly detect them and are usually unable react to the infection. For this reason, rootkit detection is a vital task for protecting Windows and it is then fundamental to make it as effective as possible.

1.1 Motivation

Current monitoring approaches cannot provide an adequate level of protection against rootkits targeting Windows OSes. In fact, most solutions operate at the same level as rootkits do [8, 22, 28]. By tampering with the functions leveraged by security tools, rootkits are able to evade detection from within the OS. Hence, anti-rootkit tools working at the OS level cannot be trusted in case of rootkit infection. When the OS is running in a virtual machine, however, this problem can be addressed in a different way. Such a scenario, in fact, allows an external observation of the OS, from a more trustworthy and isolated environment. This capability is provided by the hypervisor, which can directly access VM components without leveraging OS functions.

Such a capability enables the adoption of virtual machine introspection (VMI [14, 21]), which consists of inferring the guest OS semantics from the analysis of the status of VM components. VMI provides a valuable tool to counter rootkits since they can hardly conceal their presence to a monitoring system not dependent on OS functions.

On the one hand, VMI on Windows guest is however hard in practice, as it requires some specific OS information in order to make sense out of raw machine data [14]. This is one of the challenges of our present work, and it is also one of the main contribution of this work. On the other hand, VMI can be supported by the use of forensic memory analysis (FMA), which provides the means for extracting OS information from raw memory data. In fact, as stated above, since modern rootkits manipulate memory to avoid detection, they can be identified by inspecting the same memory contents [19]. This is a clear advantage over rootkits and allows the implementation of more reliable security systems.

Moreover, once the infection has been identified, the hypervisor also allows an effective reaction. In fact, by leveraging unfettered access to physical resources, a security tool can directly manipulate the VM and stop rootkit activities.

All these features, render the hypervisor a very attractive place where to implement security functionalities. In this work, we will leverage advanced VMI and FMA to help securing Windows OSes in virtualized environments.

1.2 Contribution

This work introduces and discusses a novel effective approach for countering rootkits on a Windows OS running in a VM. The implemented security monitor is external to the target machine, similarly to some recent literature [13, 35]. By leveraging VMI and current FMA tools, we developed a novel *Hypervisor-Based Introspection System* (HyBIS) for protecting a Windows OS from stealth malware, in particular from rootkits.

The proposed system extends the hypervisor to monitor the state of the running machine, to detect rootkits, and to react to the discovered anomalies. The monitoring functionality leverages VMI techniques to infer the guest OS status. In order to detect rootkits, guest memory is scanned for kernel objects which may have been hidden. Such a scan is performed on memory dump files

by means of FMA techniques and tools. Although such tools are typically used for offline analyses, the proposed system utilises them in a live way, during the system execution. To this purpose, HyBIS provides a novel dumping system which allows improving the performance of the memory acquisition task. Furthermore, a novel reaction approach is implemented that makes use of the hypervisor to manipulate memory contents while the virtual machine is running. This capability is leveraged to prevent the execution of detected rootkit processes. HyBIS allows detecting and reacting to rootkits effectively on recent Windows OSes. HyBIS successfully proves that the combination of VMI and FMA provides a valuable tool for countering rootkits on Windows OSes.

2 Related Work

This section surveys most relevant related work and stresses the main differences with respect to our solution.

Zhang et al. [33] leverage SMM, an advanced x86 execution mode, for detecting memory-based stealthy malware. Their SPECTRE framework can introspect a live operating system and supports both Windows and Linux OSes. However, this framework is vulnerable to hardware-based attacks, such as [34]. Furthermore, their work is limited to Windows XP SP3.

In [16], Hizver and Chiueh make use of Volatility for the analysis of VM execution states. Their RTKDMS system is able to perform real-time monitoring at the hypervisor level. Differently from HyBIS, such an architecture leverages an additional VM for the introspection analysis. Again, the experiments are limited to Windows XP. Furthermore, their system does not tackle the rootkit threat specifically, neither it explores any reaction possibility.

Deng et al. [9] propose the SPIDER architecture, a stealthy program instrumentation and debugging framework built upon hardware virtualization. SPIDER enables monitoring memory read/write at any address. Nonetheless, unlike HyBIS, it requires an in-guest agent which modifies the guest OS kernel.

In [20], Lengyel et al. describe DRAKVUF, a novel dynamic malware analysis system based on Xen, which improves hardware resources usage efficiency. DRAKVUF takes advantage of the hardware virtualization extensions to provide a transparent and scalable environment to enable in-depth analysis of malware samples. In this case, the target OS is Windows 7 SP1 in both 32- and 64-bit versions.

Differently from DECAF [15] which only supports introspection of 32 bit guests, our HyBIS approach also allows inspecting 64 bit Windows guests. Furthermore, Henderson et al. declare that VMI has limitations against some types of attack (due to the ability of the guest attacker to modify memory contents at will). While we agree that memory analysis can be circumvented in theory, we believe in practice continuous monitoring and effective/smart diff-deltas can render an attack much less practical.

As regards memory forensics issues, an interesting work by Balzarotti et al. [7] assesses the impact of GPU-assisted malware [10] on memory forensics. They

discuss different techniques that malware can adopt to hide its presence on GPUs. Their analysis shows that, by offloading some computation to the GPUs, it is possible to successfully hide some malicious behavior.

Harrison [13] suggests an approach that is somewhat similar to HyBIS. However it aims to be integrated into other IDS solutions [11], and mostly focuses on the analysis phase. HyBIS, instead also explores the novel reaction capabilities given by the combination of VMI and FMA techniques.

3 HyBIS - Approach and Functionalities

This section describes the approach behind HyBIS, the proposed Hypervisor-Based Introspection System for Windows. HyBIS combines FMA techniques with the VMI approach. As mentioned above, the main goal of our work is to improve Windows security in virtualized environments. In particular, our research focuses on protecting such OS from rootkits. In order to protect Windows from modern rootkits, we mostly focused our studies on the RAM component. Memory, in fact, stores both code and data and is involved in almost every operation performed on the machine during OS execution. Thus, RAM can be considered the most complete source of information about the status of a running OS at a specific time. For such a reason, modern rootkits use to manipulate memory to conceal their activities and resources. Nonetheless they reside in RAM while running, thus giving the opportunity to detect their presence. Hence, memory is the best place where to look for inferring the current status of the target machine. FMA enables performing such a task in an effective and convenient way.

The basic idea we followed for the development of our security system was that the hypervisor can do more than what it is intended for. The chosen design approach was then to augment the hypervisor capabilities by means of introspection techniques. We extended the hypervisor by introducing the following functionalities: i) *Monitoring*: the hypervisor is enabled to monitor the machine state in order to realize if something anomalous is happening; ii) *Analysis*: the hypervisor is enabled to analyze the state of the guest OS in order to detect the presence of rootkits; iii) *Reaction*: the hypervisor is enabled to react when a rootkit is detected and block its activities. The above functionalities leverage internal hypervisor functions as well as external libraries and tools. The internal functions provide direct access to virtual machine hardware components. In particular, they allow to monitor the VM CPU and physical memory. By checking the CPU state and reading the memory contents, it is possible to implement a transparent monitoring function. Furthermore, the write access to memory can give the ability to perform changes into a running VM.

By making use of external tools and libraries, the hypervisor can be given even more capabilities. For instance, by integrating memory forensic functions, it is possible to implement advanced analysis techniques, which may allow detecting the presence of rootkits into the system.

Monitoring - Checking The System State: the virtual machine state can be analyzed by means of VMI. As stated before, the hypervisor has the

ability to access virtual hardware resources directly, allowing the monitoring of all the VM components. In particular, we chose to monitor CPU and memory as they are core components of the machine.

CPU state changes can be easily monitored by using internal hypervisor functions. Such functions allow, for instance, checking the current operating mode, or inspecting registers.

VM memory contents changes can be monitored by means of differential dumps. With this approach, memory dumps are periodically generated to check if a particular area has been modified. In order to implement such a functionality, an initial memory snapshot must be taken at a specific time. Such a snapshot can then be used as basis for the comparison with the following checkpoints.

Analysis - Detecting Rootkits: as previously explained, modern rootkits are able to manipulate memory objects at runtime to conceal their activities. Hence, in order to detect rootkits, the analysis functionality should focus on the memory contents. This kind of analysis requires advanced forensic techniques to discover an infection. A convenient approach would be then to make use of functions from an external forensic tool or library.

Memory forensic analyses are commonly based on memory dumps. Hence, a memory acquisition functionality is required to make use of the forensic tools. Fortunately, most hypervisors implement their own dumping facility which can be used to acquire guest memory. Such a facility can be easily expanded/adapted to improve the acquisition process and realize the above-mentioned differential dumping functionality.

Reaction - Countering Rootkits: in order to react when a rootkit is detected, some kind of action has to be taken to prevent it from performing further activities. As such, guest memory can be used to implement the reaction functionality.

Since rootkits, even if concealed, reside in memory, it would be a good approach to counter them into the same place. Once again, hypervisor functions can come in handy: by writing into guest memory, it could be possible to delete the detected rootkit from memory, or to block its current execution.

3.1 HyBIS Functionalities

We set up to implement four high-level functionalities in HyBIS: Automatic boot dump generation; Smart differential dumping; Detection of hidden rootkit processes; Blocking of hidden rootkit processes. In the following, we detail each functionality.

Automatic Boot Dump Generation: as explained above, the monitoring functionality should allow deciding when an analysis operation would be appropriate. Since the analysis functionality operates over memory dumps, we decided to automatically generate a dump on the basis of some hardware event. In particular, we chose to monitor the VM during the boot phase in order to produce a dump at the very beginning of the Windows loading procedure.

This choice has a twofold reason. Firstly, it aims at determining the first feasible moment for analyzing a memory dump with a forensic tool; in fact, these

tools need the kernel to be loaded in order to work. Secondly, such initial dump can be used as starting point for a following monitoring of the memory; in fact, after the kernel has been loaded, most of the system areas remain fixed during the rest of OS execution.

Hence, this function should allow HyBIS to automatically generate a memory dump as soon as the Windows kernel process starts.

This objective has been chosen to demonstrate how, by means of introspection, the VM state monitoring can be effectively used for determining meaningful moments of the OS execution.

Smart Differential Dumping: besides the CPU, the monitoring functionality can check specific virtual machine memory areas for changes, in order to decide if an analysis operation is needed.

As stated above, a differential approach can be taken to perform this kind of monitoring. However, memory acquisition can be a very onerous task to perform, especially when it has to be repeated over time. So, it is important to do such an operation efficiently in order to not compromise the guest system performance. Since only some memory ranges need to be checked, there is no need to dump the whole memory at every checkpoint. Instead, it should be enough to acquire only the ranges we are interested in.

This function should allow HyBIS to update a previously created dump by acquiring selective ranges and overwriting them into the corresponding areas. Previous contents of such ranges should be backed up in separate files in order to allow the comparison between different checkpoints. With such an approach, it can be said that HyBIS uses "dynamic" dumps.

Dynamic dumps can also be used to improve the acquisition process necessary for the forensic analyses. In fact, since such analyses usually involve only some ranges of the whole dump, it is possible to use the update mechanisms described above, to perform the analyses of different checkpoints without needing to create multiple dumps of the whole memory.

This objective has been chosen to demonstrate how monitoring the VM memory can be both effective and efficient.

Detection of hidden rootkit processes: it is well-known that rootkits try to hide their processes to avoid detection. This is effectively obtained by implementing the DKOM technique [17]. In particular, a rootkit that wants to hide a process, could remove the corresponding object from the active process list. In fact, Windows uses two lists of processes: one for scheduling, and one for tracking. A process whose object is removed from the tracking list, will be invisible while still active. As such, hidden processes can be detected by means of a cross-view analysis [5]. More specifically, this can be done by scanning memory for process objects, and comparing the results with the active process list. If a scanned process is not present in this list it is likely to be a hidden rootkit

[5] It is worth noticing that malware able to infiltrate the task scheduling could be executed without being on process lists. Current HyBIS does not test for this. This is left as future work.

process. This function should allow HyBIS to detect hidden processes by creating a memory dump and scanning it for concealed process objects.

This objective has been chosen to demonstrate how FMA can help in detecting rootkits on running guest OSes.

Blocking of hidden rootkit processes: once a hidden process has been detected, it should be blocked to prevent it from keeping performing malicious activities. This action will not clean the infection but it could be a first step to defeat the rootkit.

A good idea for blocking a hidden process would be to exclude it from scheduling, thus preventing its execution. This can be done using the DKOM technique in a similar way as that used by rootkits. More specifically, we can block a hidden process by removing the corresponding object from the scheduling list. This function should allow HyBIS to manipulate the VM memory in order to prevent the rootkit process to be executed.

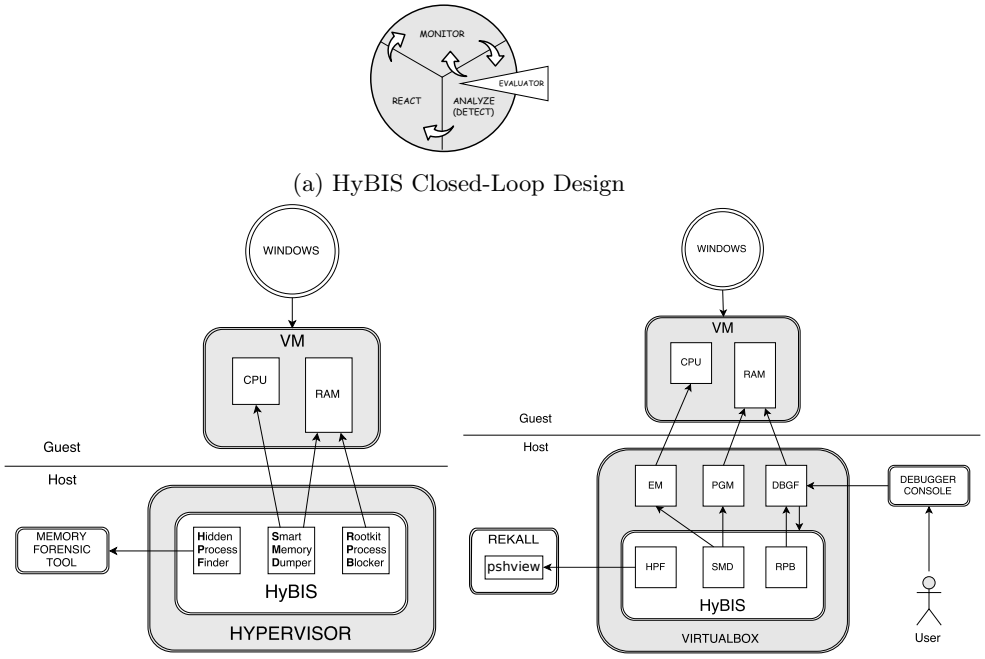
This objective has been chosen to demonstrate how the hypervisor capabilities allow an effective reaction to a rootkit infection by means of memory manipulation.

4 HyBIS - Design, Architecture and Implementation

This section shows how HyBIS was designed to extend the hypervisor capabilities for securing a guest Windows OS from the rootkit threat. First, we show the overall HyBIS operation from a high-level point of view. Next, we describe the HyBIS architecture. Finally, we describe and motivate the technologies chosen to build up the first HyBIS prototype, and the most relevant implementation details.

The high-level overview of the HyBIS operating mode is depicted in Figure 1a. The new functionalities are designed to work as a closed loop control system. The monitoring phase extracts information from a running machine and intercepts events which could reveal the presence of rootkits. The analysis phase examines the system in order to evaluate if a rootkit infection occurred. In such a case, it triggers the reaction phase, otherwise it returns to the monitoring phase. The reaction phase tries to remove the infection or block the rootkit for preventing further malicious activities.

The monitoring and reaction functionalities leverage the introspection and control capabilities of the hypervisor. The analysis functionality is based on forensic functions provided by external tools but needs some additional intervention to interpret the results and taking further actions. Since the complexity of such a task, some kind of intelligence is needed to take decisions. This is represented by the *evaluator*, which is an external component that can be inserted into the analysis phase. The evaluator functionality can be performed by a human examiner as well as an external plugin which implements advanced AI techniques, such as Machine Learning, Expert Systems, Human Expertise, and so on.



(a) HyBIS Closed-Loop Design

(b) HyBIS Architecture

(c) The HyBIS prototype leverages the VirtualBox's Execution Manager (EM), Page Manager (PGM), and Debug Facility (DBGF) components

Fig. 1: HyBIS Design, Architecture, and Implementation Details

4.1 Architecture

The HyBIS architecture is shown in Figure 1b. As can be seen, on the guest side, there is a Windows OS running on a virtual machine (VM). On the host side, there is the hypervisor that controls the VM, which incorporates the HyBIS component. The HyBIS component extends the hypervisor with the new security functionalities. Such functionalities are implemented by three components, described below. Outside the hypervisor, on the same side, there is the memory forensic tool, which is used by HyBIS to provide advanced analysis capabilities.

HyBIS Components: HyBIS includes three components, running in parallel with the guest OS, without interrupting/suspending its execution:

- *Smart Memory Dumper* (SMD): this component allows creating the dynamic dumps described in the previous section; it leverages the hypervisor to read the VM's RAM contents and create or update dump files on the host disk;
- *Hidden Process Finder* (HPF): this component allows detecting hidden processes running in the guest Windows OS; it leverages the external memory forensic tool to perform analyses on the dump files created by SMD;

– *Rootkit Process Blocker* (RPB): this component allows blocking a detected rootkit process on the guest OS by preventing it from being scheduled for execution; it receives from HPF the information on the detected process and leverages the hypervisor for manipulating the VM's RAM.

4.2 Technology Info

In this section we discuss the technologies selected for the development of the first HyBIS prototype. In particular, the target Windows version, the hypervisor and the forensic tool, have been chosen due to their effectiveness and wide deployment base.

The Target OS: most of the latest security-related work still focuses on Windows XP or Windows 7 OSes. However, Windows 8 introduced some internal changes (such as [18]) and security mechanisms (see [27]) which partially invalidate previous results. For instance, the removal of the `KiFastSystemCall` function makes all rootkit techniques based on this function unusable [26]. Furthermore, the latest Windows 10 OS appears to keep such changes, rendering previous work yet more obsolete.

At the time we started the development of HyBIS, Windows 10 was only available in its Technical Preview release. Hence, we selected Windows 8.1 (which is much more widespread then Windows 8) as the target of our experiments.

We initially decided to focus our tests on the 32-bit version since it is more efficient when performing extensive memory-related experiments. Furthermore, the 64-bit version which implements more advanced security mechanisms, would have limited our malware testbed. As such, extensive experimental activity on 64-bit OSes will be the target of future work.

The Hypervisor: most of the recent projects targeting Windows as guest OS, involve the qemu-kvm [1] or the Xen hypervisors [4]. Although these ones represent valid tools, we decided to make use of the VirtualBox [24] hypervisor. In fact, VirtualBox, apart from being one of the mainstream virtualization technologies, has two main advantages over qemu-kvm and Xen: firstly, it fully supports all Windows versions, including the latest Windows 10; secondly, it includes various VM-debugging functionalities, that allow controlling and manipulating VM components [24]. Such funcionalities can be very useful when implementating advanced introspection techniques. For the HyBIS prototype implementation the latest VirtualBox 5.0 version has been used.

The Memory Forensic Tool: among the available FMA tools, Volatility [3] is certainly the most widespread. It has a vast number of functionalities and it can count on a very active community. Nonetheless, it does not fully support all Windows kernel versions. In addition, its performance on memory analysis is quite low for our real-time usage requirement.

A derived project, named Rekall [2], overcomes these limits, while maintaining main Volatility features and advantages. Its novel kernel profiling system, enables Rekall to automatically upgrade its compatibility with new Windows versions [5]. Furthermore, thanks to some improved memory-scanning functions, it shows better analysis performances. This renders Rekall both more efficient and effective

than Volatility. Moreover, it can be integrated as part of other software as a library. Finally, since Rekall is implemented in the Python language, it can be easily installed into a variety of host OSes (This is also an advantage of VBox over Kvm and Xen). For the HyBIS prototype implementation we made use of Rekall 1.4.

4.3 Implementation Details

We will now describe how the proposed architecture was implemented by making use of the previously-mentioned technologies. The HyBIS implementation details are depicted in Figure 1c. As before, on the guest side we have a Windows OS, running in a VM. On the host side, we have the VirtualBox hypervisor, with its internal components: the Execution Manager (EM), the Page Manager (PGM), and the Debug Facility (DBGF). The HyBIS component is implemented as a new VirtualBox component, interacting with the other ones to perform its tasks. In particular, the SMD component leverages PGM for the memory acquisition, and EM for the automatic boot dump generation. The RPB component uses DBGF to manipulate guest memory. The HPF component makes use of the external Rekall component to perform advanced analyses over memory dumps.

The interaction with the HyBIS components is provided through a set of new commands on the VirtualBox integrated Debug Console. These commands will be described later in this section.

Extending VirtualBox The following VirtualBox components have been involved for the HyBIS implementation:

- Virtual Machine Monitor (`VMM`): this is the core hypervisor;
- Execution Manager (`EM`): it controls the execution of guest code;
- Page Manager (`PGM`): it controls guest memory paging;
- Debug Facility (`DBGF`): it provides a built-in debugger for the VM.

Most of the VM-related components are implemented in the VMM section. Therefore, this is the most suitable place where to insert the new HyBIS component. The EM component is leveraged to monitor the CPU operation mode in order to automatically generate the boot dump. The PGM provides all guest memory management functions and is used to implement the acquisition functionalities. The DBGF provides a lot of useful debugging functions, which are used for implementing the reaction functionality. Furthermore, it provides the console devoted to the interaction with the HyBIS component.

Integrating Rekall Since Rekall is written in Python, an interpreter must be present on the host system. For incorporating Rekall, it has been necessary to import the Python C++ library into the VirtualBox source code. The Rekall functionalities, instead, can be used by means of the provided API library, as mentioned in the previous section. In order to make use of the forensic analysis functions, a suitable session has to be set up. A Rekall session represents a specific combination of a dump file and a selected profile. When starting a session, if a valid profile for the current kernel version is found, every compatible Rekall plugin automatically becomes available to be used for a forensic analysis.

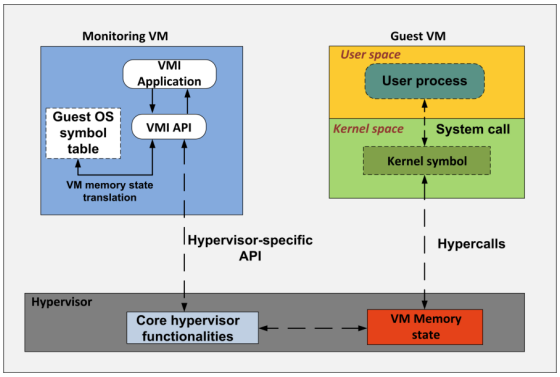


Fig. 2: A Typical VMI Architecture [30]

4.4 Further Details

HyBIS embodies a set of novel architectural and implementation-related features. In the following, a broad view and some relevant details are given.

Technically speaking, HyBIS is an hypervisor-based IDS which leverages Virtual Machine Introspection (VMI) to monitor a virtualized environment and exploits Forensic Memory Analysis (FMA) to bridge the semantic gap.

With respect to previous solutions, HyBIS uses a novel approach, whose differences are explained in the following.

Use of VMI: typical hypervisor-based IDSs reside on top of the hypervisor, in a Secure Virtual Machine (SVM) (see Figure 2).

Such an approach has the advantage of isolating the security functions so to avoid corruption. However, in such a case, the IDS requires an additional VM, and its capabilities are limited by the functionalities provided through the hypervisor APIs. HyBIS, instead, is integrated into the hypervisor, as shown in Figure 1b. As such, it is able to exploit all the hypervisor capabilities in order to have full control over the target VM.

An alternative VMI approach leverages advanced CPU features to interpose the security functions between the OS and the hardware [35]. However, this kind of hypervisor does not have the control over all the machine components. Furthermore, this approach usually requires an in-guest agent to be installed into the OS kernel. Conversely, HyBIS does not require any addition to the guest OS kernel and does not rely on any CPU feature.

So, while current VMI-based solutions have limited possibilities to exploit the hypervisor capabilities, the HyBIS novel approach gives the ability to fully exploit them by integrating into the hypervisor. By working from below the VM, it has a full overview of the whole machine state, while still being isolated from the target machine, thus avoiding both OS-level and hardware-level attacks.

Use of Memory Forensics: FMA can be a valuable means for VMI-based systems to bridge the semantic gap. However, it has not yet been fully leveraged by current solutions. This is probably due to the slowness of the memory acquisition

process. In fact, FMA tools usually operate on offline dump files, whose creation may take too long for a practical real-time usage. As such, memory acquisition, represents a critical step for the implementation of a real-time FMA-based solution. Furthermore, the soundness of the acquisition process is another serious concern, given that memory is usually dumped by tools installed into the guest OS (see [29]). Such tools have the drawback of altering memory contents and to be vulnerable to OS-level attacks which may corrupt acquired data. Alternatively, memory acquisition can be hardware-based, using techniques, such as DMA, that allow bypassing the OS. Such techniques, however, can still be bypassed by hardware-level rootkits, such as in [25] and [34], which are able to alter acquired data for hiding their presence.

By leveraging the hypervisor, HyBIS is able to acquire memory using a novel dynamic approach, without being vulnerable to OS-level and hardware-level attacks. The chosen technologies for present HyBIS implementation was not only chosen for their useful features. All of them represent an element of novelty for security research. First of all, the chosen Windows kernel version has been poorly explored in previous work. Most recent Windows-related papers still refer to Windows 7 as the subject of their studies, or as the target for their experiments. Actually, we have not been able to find any kernel-related research on Windows 8 and Windows 8.1. Instead, in our HyBIS implementation and testing, Windows 8.1 was used.

Leveraging the Virtualbox hypervisor represents another relevant contribution of this paper. Most security and virtualization studies involved other common hypervisors, like KVM or Xen. Instead, besides a few performance analyses, we were able to find only a single work involving VirtualBox for its implementation [23]. In the development of HyBIS, VirtualBox has proved to be a great tool for our purposes. In fact, its features and functionalities have been very helpful for the exploration of VMI techniques.

A similar discussion can be made about the Rekall forensic tool. To the best of our knowledge, Rekall was rarely used as part of a security research project. The only online reference found was in [6], where Rekall was used to analyze Windows profiling. This is probably due to its relatively recent introduction (2007). Most studies and researches involved Volatility instead, from which Rekall was derived. In fact, Volatility has the advantage of being more widespread and supported. However, as previously described, Rekall presents almost the same features but introduce novel features that drastically improves performance and usability. As such, we found it a valuable tool for memory forensic research.

5 Evaluation

Experiments have been performed, testing the implemented prototype effectiveness, and the functionality of each HyBIS component described in Section 3.1. Some details of the evaluation are given below.

Boot Dump Generation: As previously described, this function should generate a memory dump as soon as the Window kernel has been loaded and

begins to run. In order to prove the effectiveness of this function, the generated dump has been analyzed with Rekall. The analysis showed that the only running process found in memory was `System`, that is the Windows kernel image process. This proves that the dump generation actually occurs at the beginning of the Windows kernel execution. This dump also proved to be acquired in the very first moment the kernel loads. Dumps generated before this point failed to be analyzed by both Rekall and Volatility.

As for what stated above, we can claim that this function generates a dump at the very first suitable time for being analyzed by these kind of FMA tools. Moreover, we believe this functionality, if properly extended, can be helpful in detecting those rootkits that load during the very early stages of the Windows boot.

Dynamic Dumping: This function has been devised to explore new ways of making the dump creation phase faster for monitoring purposes. In order to test it, we identified a restricted memory area to update a previously created dump, while still allowing forensic analysis functionalities. In particular, we selected a range of 250MB, which empirically showed to always contain all kernel process objects. ASLR [12], does not to affect the results of our tests. In fact, the leveraged tools are able to find out the correct position for the kernel structures.

With the purpose of proving the usefulness of this functionality we first created a dump, then started a new process in the guest, and eventually updated the dump by acquiring the above mentioned memory range. Thereafter, we scanned the updated dump with Rekall, to retrieve the active processes list. As expected, the new process was correctly present in the list.

This demontrates that such a reduced memory acquisition speeds up the acquisition process, while not preventing Rekall to properly work, with respect to specific operations. This enables a new form of memory monitoring, which is not limited to single page changes, but involves larger areas. In fact, by leveraging FMA, it is possible to check memory for changes in the OS-level data (e.g. the presence a new process).

Detection and Blocking of Hidden Processes: In order to monitor guest OSes, HyBIS creates a monitoring thread for each VM. This thread periodically dumps the VM memory and analyze it with Rekall. In particular, we created an improved version of the `psxview` plugin, called `pshview`, in such a way that it could be automatically filter the suspected hidden processes. Specifically, it tries to distinguish among dead processes, which may remain in memory for a while after their termination, from those which are still present in the system lists, such as the thread list or the session list. In addition, this function avoid scanning the whole memory, as this seldomly proved to discover more processes than those found by reconstructing kernel object lists. The user also has the ability to enable or disable the scan function, as well as increment or decrement the scanning rate.

If after the analysis the `pshview` plugin returns a suspected hidden process, a pop-up message will be shown to the user, warning about the detected threat, and suggesting to block it through the `.psblock` function. This removes the

hidden process object from the system scheduling list (as well as other system lists), thus preventing its execution. In order to prove the effectiveness of this action, we again dumped memory and fully analyzed it to verify the blocked process was not among the active processes lists anymore.

Testbed Setup: All experiments were conducted on a machine with 12 GB of memory and CPU Intel Core i7 2.4GHz. Each VM had 1 GB of memory (common size on AWS), and Intel VT-x, EPT, and PAE features enabled. As stated above, the target OS was Windows 8.1 32-bit. The OS was infected with different rootkit specimens, chosen among the most widespread and dangerous, such as ZeuS [31] and ZeroAccess [32].

6 Conclusion and Future Work

In this work we shed light on some security issues of the Windows OS. Our main contribution is the design of the novel HyBIS architecture, which successfully combines VMI and FMA to build up an anti-rootkit security system for Windows. VMI is used to examine the Windows status by means of hardware monitoring, while FMA is used to carve meaningful information from raw memory data. A rich experimental campaign was performed over most relevant malware specimens, allowing us to detect and block different hidden malicious processes. Given the architectural complexity of the Windows kernel security field, the results reported in this paper — other than being interesting on their own — also pave the way for further research.

References

1. Kvm. http://www.linux-kvm.org. [Online; accessed 20-Feb-2017].
2. Rekall memory forensic framework. http://www.rekall-forensic.com. [Online; accessed 20-Feb-2017].
3. The volatilty foundation. http://www.volatilityfoundation.org. [Online; accessed 20-Feb-2017].
4. The xen project. http://xenproject.org. [Online; accessed 20-Feb-2017].
5. Rekall profiles. http://rekall-forensic.blogspot.it/2014/02/rekall-profiles.html, feb 2014. [Online; accessed 20-Feb-2017].
6. Windows Virtual Address Translation. http://www.rekall-forensic.com/posts/2015-08-03-address_translation.html, 2015. [Online; accessed 20-Feb-2017].
7. D. Balzarotti, R. Di Pietro, and A. Villani. The impact of GPU-assisted malware on memory forensics. *Digit. Investig.*, 14(S1):S16–S24, aug 2015.
8. R. Battistoni, E. Gabrielli, and L. V. Mancini. *A Host Intrusion Prevention System for Windows Operating Systems*, pages 352–368. Springer Berlin, 2004.
9. Z. Deng, X. Zhang, and D. Xu. Spider: Stealthy binary program instrumentation and debugging via hardware virtualization. In *Proc. 29th Annual Comp. Sec. Applications Conf.*, ACSAC '13, pages 289–298, New York, NY, USA, 2013. ACM.
10. R. Di Pietro, F. Lombardi, and A. Villani. CUDA Leaks: A Detailed Hack for CUDA and a (Partial) Fix. *ACM Trans. Embed. Comput. Syst.*, 15(1):15:1–15:25, Jan. 2016.
11. R. Di Pietro and L. V. Mancini. *Intrusion Detection Systems*. Springer Publishing Company, Incorporated, 1 edition, 2008.
12. Y. Gu and Z. Lin. Derandomizing kernel address space layout for memory introspection and forensics. In *Proc. 6th Conf. on Data and Application Security and Privacy*, CODASPY '16, pages 62–72, New York, NY, USA, 2016. ACM.

13. C. B. Harrison. *ODinn: An In-Vivo Hypervisor-based Intrusion Detection System for the Cloud*. PhD thesis, Auburn University, 2014.
14. Y. Hebbal, S. Laniepce, and J.-M. Menaud. Virtual machine introspection: Techniques and applications. In *10th Intl. Conf. on Availability, Reliability and Security (ARES)*, pages 676–685, Aug 2015.
15. 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 *Proc. International Symposium on Software Testing and Analysis*, ISSTA 2014, pages 248–258, New York, NY, USA, 2014. ACM.
16. J. Hizver and T.-c. Chiueh. Real-time deep virtual machine introspection and its applications. In *ACM SIGPLAN Notices*, volume 49, pages 3–14. ACM, 2014.
17. G. Hoglund and J. Butler. *Rootkits: Subverting the Windows Kernel*. Addison-Wesley Professional, 2005.
18. A. Ionescu. How control flow guard drastically caused windows 8.1 address space and behavior changes. http://www.alex-ionescu.com/?p=246, 2015. [Online; accessed 20-Feb-2017].
19. J. D. Kornblum. Exploiting the rootkit paradox with windows memory analysis. *International Journal of Digital Evidence*, 5(1), 2006.
20. 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 *Proc. 30th Annual Computer Security Applications Conference*, 2014.
21. F. Lombardi and R. Di Pietro. Secure virtualization for cloud computing. *Journal of Network and Computer Applications*, 34(4):1113 – 1122, 2011.
22. C. Mahapatra and S. Selvakumar. An online cross view difference and behavior based kernel rootkit detector. *SIGSOFT Softw. Eng. Notes*, 36(4):1–9, Aug. 2011.
23. D. Mulfari, A. Celesti, A. Puliafito, and M. Villari. How cloud computing can support on-demand assistive services. In *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility*, W4A '13, pages 27:1–27:4, New York, NY, USA, 2013. ACM.
24. Oracle Corp. Oracle vm virtualbox programming guide and reference. http://download.virtualbox.org/virtualbox/SDKRef.pdf, 2016. [Online; accessed 20-Feb-2017].
25. J. Rutkowska. Beyond the CPU: Defeating Hardware-based RAM acquisition. *Black Hat Briefings*, 2006. [Online; accessed 20-Feb-2017].
26. M. Tech. Intercepting all system calls by hooking kifastsystemcall. http://www.malwaretech.com/2015/04/intercepting-all-system-calls-by.html, apr 2015. [Online; accessed 20-Feb-2017].
27. M. TechNet. What's changed in security technologies in windows 8.1. https://technet.microsoft.com/it-it/library/dn344918.aspx, jul 2013. [Online; accessed 20-Feb-2017].
28. W. Tsaur and L. Yeh. Identifying rootkit infections using a new windows hidden-driver-based rootkit. In *International Conference on Security and Management, Las Vegas, USA*, pages 16–19, July 2012.
29. S. Vömel and F. C. Freiling. Correctness, atomicity, and integrity: Defining criteria for forensically-sound memory acquisition. *Digital Investigation*, 9(2):125 – 137, 2012.
30. T. Y. Win, H. Tianfield, Q. Mair, T. A. Said, and O. F. Rana. Virtual machine introspection. In *Proceedings of the 7th International Conference on Security of Information and Networks*, SIN '14, pages 405:405–405:410, New York, NY, USA, 2014. ACM.
31. J. Wyke. What is Zeus? Technical report, Sophos, may 2011.
32. J. Wyke. Zeroaccess. Technical report, apr 2012.
33. F. Zhang, K. Leach, K. Sun, and A. Stavrou. SPECTRE: A dependable introspection framework via System Management Mode. In *Proc. 43rd IEEE/IFIP Intl. Conf. on Dependable Systems and Networks (DSN)*, DSN '13, pages 1–12, Washington, DC, USA, 2013. IEEE Computer Society.
34. N. Zhang, K. Sun, W. Lou, Y. T. Hou, and S. Jajodia. Now you see me: Hide and seek in physical address space. In *Proc. 10th Symp. on Inf., Comp. and Comm. Security*, ASIACCS '15, pages 321–331, New York, NY, USA, 2015. ACM.
35. X. Zhong, C. Xiang, M. Yu, Z. Qi, and H. Guan. A virtualization based monitoring system for mini-intrusive live forensics. *Int. J. Parallel Program.*, 43(3):455–471, June 2015.