Lokke, a hybrid security hypervisor

Otávio A. A. Silva[1], **Paulo Lício de Geus**[1]

[1] Universidade Estadual de Campinas (Unicamp)

otavios@lasca.ic.unicamp.br , paulo@lasca.ic.unicamp.br

Abstract. *This work did ample research on techniques used by advanced threats that aim to evade detection systems, elevate privileges and manipulate objects in a modern OS kernel, using the Windows 10 kernel as a test bench. Given state-of-the-art attacks in kernelspace, this work's main goal is to design a secure mechanism to protect the OS kernel against a class of attacks, not relying upon any specific vector. This mechanism is based on hybrid virtualization and combines the advantages of Type 1 and 2 hypervisors, where the hypervisor runs at the same level as the OS kernel does, but within a privileged execution framework. The design of this security framework allows for the integration with other security subsystems, by providing security policies enforced by the hypervisor and independently of the kernel.*

1. Introduction

Modern OS kernels are built with security as a moving concept, where many approach were made to decrease the kernel overhead, enhance user-level switch and performance, with security policies integrated as best practices or bug fixing. This introduced in many aspects behaviors which were exploited by attacks in the fashion of OS design, as the first wildly use of buffer overflow attack, by the Morris Worm [22], or many user privileged elevations in Windows User Account Control[7]. Although, over the years, these attacks demanded changes in the design concept of any modern OS, as in dynamic libraries, process, and memory organization, most sophisticated attacks nowadays are still abusing in some fashion the behave of the OS to provide userspace services. These exploited behaviors are unlikely to be patched or change any time soon by being intrinsic in how the OS is built, and by the high attacks sophistication demanded, usually named Advanced Threat.

Advanced Threats usually elevate to, or resides, in the kernel environment, using sophisticated attack vectors to evade threat detection tools—which usually operate on the kernel side—and obtain privileged execution while controlling the kernel. Those threats abuse the OS behavior, usually manipulating kernel objects directly or interacting with any other kernel interface in the userland. One of the most common vectors of attack, manly present in Windows, is the Direct Kernel Object Manipulation[6].

Historically, Windows is the most targeted OS by all kinds of threat scenarios and vector by its high market share and how the OS itself works. Since Windows 7, Microsoft changed the OS design and slowing have been enforcing new security approach to user-level, with Windows 10 being the most recent and sophisticated release of Windows. The main difference between Windows 10 and other OS, including its past releases, is the use o virtualization as a security layer for the kernel itself. Many subsystems were introduced as the Hypervisor-protected Code Integrity (HVCI), Credential Guard (C.G.)[21], and

Virtualization Based Security, which uses Hyper-V as smaller scale virtualization. This change in the design was not new, over the years, academic and industrial solutions were made using hypervisor as a security abstraction to the OS

Although introducing the use of virtualization into Windows 10, Microsoft focused on protecting critical parts of OS from being tampered, but it did not change any previous behavior abused by advanced threats, *e.g.*, direct kernel object manipulation. The last Microsoft Security Intelligence Report [10][1] demonstrated that the same threats from the past ten years still hitting Windows 10. With phishing attacks and botnets being the most common scenario, with 2018 settling down as a year of increasing ransomware botnets, 2019 had the lowest ransomware rate since 2016. Although highly advanced and persistent threats—usually attack resilient in the kernel side—are not present, this absence does not imply the non-occurrence of a sophisticated attack vector, it does indicate that advanced threats are not a common threat for standard users. The Threat Intelligence report [1] presented by Kaspersky evinces that highly advanced threats are becoming surprisingly common toward strategic users, *i.e* desktop users with a leading position in an organization.

Thus, Windows 10 still the most target OS with the majority of threat samples, evasiontechniques, and bad practices in the OS, mitigated in a long chain of checking inside the kernel. With that in mind, this work represents the ongoing development of a virtual machine manager as a secure framework, built to deliver particular security policies to the OS running, using Windows 10 for benchmark the proposed secure mechanism. This mechanism is based on hybrid[5] Intel Vt-X[9] virtualization, combining the advantages of Type 1 and 2 hypervisor, where the virtual machine manager runs at the same level as the OS kernel, avoiding the semantic gap[4] frequent in virtual machine(V.M.) introspection; when the OS runs in the top of the hypervisor, who needs to pause the V.M to interfere within the kernel behavior, memory accessor any other hardware event.

The use of security policies comprehends changes in the OS kernel's behavior, enforcing the best practices in some scenarios, or even blocking entering a behavior, such as avoiding any direct access into the kernel objects. This approach differs from the one used by both academic and industrial solutions, by seamlessly interfering in a kernel subsystem to avoid or change behavior by the hypervisor perspective. The main objective of these changes is to provide a security policy already existent and validated in a given subsystem of an OS within another OS For instance, the Discretionary Access Control (DACL) used in kernel objects on many Unix OS are not used in Windows by default[18], but a hypervisor can easily enforce it without changing Windows's kernel or breaking retro-compatibility of kernel drivers; which is the reason Windows does not enforce it by default.

Although the core of the hypervisor is made independently of any OS kernel, using the Intel Vt-X instruction set and how the processor architecture works to the OS kernel, its policies handlers must be made exclusively to a given OS; in this stage of development, its made only to Windows 10 kernel interface. Besides the possibility of applying security policies absent in Windows, the choice to focus on developing the handlers first for Windows is because of the vast universe of attacktechniques and samples available.

[1] From March 2017 to February 2019

Thus, one of the first stages of this work started by researching the most critical and inherent attack superficies of Windows 10, searching approaches capable of manipulating Windows privileged objects, while being able of work undetectable by anti-virus engines or any security framework available in Windows, relying exclusively on its features or in the behavior of kernel interfaces. This stage's first results were kernel attacks through a device driver's object, resulting in the physical memory access by an unprivileged user. These attacks were reported and received CVE[2] numbers 2018-8060 and 8061 [27]. This result was used in combination with public kernel and third-party drivers exploits, to ensure that the protection is empirically working for real-world threats, including ones previously unknown.

The rest of the paper is organized as follows. Section 2 describes techniques used to bypass and manipulate the protection system, using only Windows 10 features. This section also contains an approach for Direct Kernel Object Manipulation (DKOM). Section 3 presents the state of the art of security hypervisors capable of reducing part the attack surface described in Section 2. In Section 4, the security framework and its hypervisor is described. Section 5 includes the achieved results on this development stage, by mitigate a widely class of attacks. Finally, we conclude the paper in Section 6, with the next steps to be made into this work..

2. Windows abuse scenario

Almost all abusing in Windows relies on some fashion into its internals, i.e., not exposed functions or services. The WinAPI is the regular programming interface that exposes features and capabilities available to a specific version of Windows. It is separated into three architectural sets, .NET API, Windows Runtime API (WinAPI), Windows Win32, and COM API. What is usually called internal API is the NtAPI, because it is the API used by all sets of the WinAPI to provide the features and services available in the userspace.

Usually, the WinAPI is just a wrapper to a NtAPI call, for example, doing the dynamic allocation of structures and handling errors in kernel standard (NtStatus) to a more readable error standard. Although the internal tag is usually referenced to functions with Nt in its name, e.g., NtClose, it can be other Windows function naming standard, as Ldr or Zw symbolic naming, e.g., *LdrRegisterDllNotification* or *ZwWriteVirtualMemory*, for the loader or kernel namespace. As an example, the WinAPI function *OpenProcess* internally uses the NtAPI call *NtOpenProcess*, needing more parameters than in the first scenario. The same functionality is available to *ZwOpenProcess* as for *NtOpenProcess* in the exported symbols for Windows kernel. The authors of the book[33] provide a more precise and complete description of Windows Internals in all three architectural sets, including the use of internal subsystems, such as I/O, storage, and memory management.

2.1. Handle hijacking and process impersonalization

Windows has compatibility and seamless user experience as an essential premise in its design, by automatically serving user's processes with a high range of functions, like translation, program auto-compatibility, and accessibility. That ensures, for instance, that even if a program does not offer any accessibility, Windows can read all of a program's text using voice service. Another example is when a program fails to run using WinAPI

[2]Common Vulnerabilities and Exposures

for Windows 10, and it can automatically run in WinAPI for Windows 7, without the user ever noticing it. This is achieved by having Windows Services monitor all user processes in execution and have multiple open handles for each process that may need intervention. These services have handles with desire access, including memory operation and thread control, which is enough to control the whole execution of a given process if necessary.

Although many vulnerabilities use flaws in this design to elevate privileges of the current user while in userspace, this is not the primary goal of this section, including attacks of copying or duplicating handles from the kernel side. A wide range of attacks that rely on these vulnerabilities is in the repository of UACME[7], including some working in Windows 10. However, this design can be abused usually but not necessary by the administrator against protected processes by third-party kernel drivers, e.g., anti-virus or anti-cheat.

The Windows process Client/Server Runtime Subsystem (CSRS.exe) and Services (services.exe) have handles to all running processes, they are executed early in the boot stage of user-side before any non-Microsoft software and are processes protected by the kernel, having Protected Process Light enabled. The process service.exe has one of its purposes to create and maintain the first instance of Services Host Process (SVCHOST.exe), which inherits its parent handles and critical permissions. Many Windows Services are built as a dynamic library and loaded by a child of the first instance of SVCHOST, but with dropped permissions. Each new instance of SVCHOST becomes a new Windows Service, in some cases running with a user's token and in others with the System token. Thus, meaning that some services in the ring of an access token from a user can be abused to manipulate a protected process in some fashion, by obtaining its opened handlers inherited by the parent service.

This approach is useful because most real-world protection works by registering kernel callbacks in special WinAPI (and all its internal) functions, interposing the call, and changing its execution. A common approach is always returning permission denied by OpenProcess, or caping its return handle to only a set of permissions, such as QueryInformation. In this scenario, if a secure browser launched by an anti-virus solution has the user's permission, any user, including the administrator, cannot open a handle to it because of the A/V capping denying the handle opening.

The given scenario can be analyzed in how the handle is managed, i.e., which callback or action is used for the protection, and when it happens. The when is important because some protection system also scans for open handlers to protected process to cap or close it, so this action happens independently of the callback. That leads the attacks in three different approaches:

- Obtaining a handle opened before the protection system registers the callbacks, e.g., handle inherited or used by Windows services.
- Obtaining a handle by a WinAPI method that the protection system does not register a callback, e.g., some Nt internal job queue or a WinAPI feature.
- Obtaining the handle and using it before a kernel filter, callback, or any event registered by the protection system being dispatched by the kernel, i.e., a race condition.

2.2. Windows kernel interface abuse

Before Windows 10 Anniversary Update, a user was able to load self-signed drivers if Windows was booted in Test Mode, but after this update, it also became necessary to disable Secure Boot. That introduced another guarantee to kernel-side: drivers unsigned by Microsoft will never load inside the chain of trust started in the UEFI from the boot. Therefore, threats started to rely upon vulnerable device drivers, duly signed by Microsoft, to manipulate the kernel by touching its objects; structs, memory pages, and even physical memory. This class of attacks is not new and is known as Direct Kernel Object Manipulation (DKOM), the authors of [6] provides a better understanding of this subject and demonstrate how rootkits are using DKOM to subvert modern OS

2.2.1. I/O Control interface abusing

There are many classes of flaws that can be exploited to obtain access to kernel objects through a vulnerable driver, like memory corruption, invalid pointer management and improperly ACL (Access Control List). Although the driver's vulnerability was the trigger used to reach the kernel-space, some of these vulnerabilities were in how an unprivileged user can directly interact with kernel interfaces in Windows if the driver's developer did not manage the ACLs correctly. Unlike others OS, e.g., Linux or BSD, where all kernel interfaces are considered privileged objects demanding special groups or capabilities, some kernel interfaces in Windows are by default accessible by a regular user. One of the most used interfaces for DKOM and other attacks is the Device Input and Output Control (IOCTL) interface, which I/O Request Packets (IRP) are sent when a particular operation occurs in the driver's device object, i.e., I/O interface.

The driver I/O interface is created with *IoCreateDevice*, and if no Discretionary Access Control List (DACL) is explicitly defined, it by default give full access (GENERIC_ALL) to the administrator, and R/W/E access [3] to every other user. The driver can use or allocate general-purpose IRP with defined Major Functions (M.F.), such as create/close or allocate a new IRP, with IoBuildDeviceIoControlRequest, to handle device control request synchronously. The descriptor structure for a driver object contains an array of function pointers, each pointing to an M.F. for dispatching and handling the driver operation from and to the user-side. It is in the dispatch function that the driver can handle each I/O control code from each IRP separately, and act according to the functions provided to userspace.

Due to the permissions available to interact with this kernel interface by an unprivileged user, attacks that abuse this interaction usually attempts to elevate the user privileges or evade protection systems from the kernel-side. The most straightforward way to achieve this is by using an IOCTL, which allows read or write on a memory page in the kernel-side, thus abusing a regular feature of the driver to operate maliciously. With that in mind, a threat has to use the device driver without exploring any specific vulnerability besides permissive and default ACL, to be able to manipulate a kernel object directly to elevate its privileges. Even out of a privileges elevation scenario, exploiting third-party drivers have the benefit of being ever vulnerable because they certificate hardly are revocated, thus continuing valid after a patch is released. So the vulnerable version can be

[3] Mask GENERIC_READ | GENERIC_WRITE | GENERIC_EXECUTE

turned into a kernel-side toolkit, where an already elevated threat can use it to manipulate the kernel by loading the vulnerable version and exploiting it.

2.3. Empirical proof of conception

Therefore, to prove that it continues to be possible to abuse IOCTL from device drivers in the last release for Windows 10[4], relying exclusively on lousy ACL and lousy security approaches by drivers, one of the most popular software for monitoring the hardware for Windows was tested: Hardware Monitor (HWMonitor x64). This software relies on a user software to display sensors, e.g., core and memory temperature and cooler frequency, and on a device driver to do all the actual reading and writing to the hardware interface. The user interface controls all the driver actions through IOCTL, from reading CPU capabilities to controlling cooling policies.

Similarly, with other drivers with ACL flaws, an unprivileged user can issue IOCTL to the driver directly, using all of its interfaces with the hardware. Reverse engineering, the device driver revealed many operations capable of manipulating CPU registers and I/O ports, allowing direct access to physical memory for reading and writing. All the flaws discovered were reported to the software maintainer, and the two most critical (CVE 2018-8060 and 8061 [27]) were reported to the Mitre Common Vulnerabilities and Exposures (CVE) after being fixed in the stable branch of HWMonitor. With this attack, an unprivileged user achieved direct access to physical memory, including access functions to read or write virtual kernel addresses, thus, having full control of all kernel space.

3. Related works

The use of a VMM (Virtual Machine Manager), also known as a hypervisor, to monitor or interfere with the execution of an operating system, became a recurrent field of research. Among the reasons, the migration of threat to the kernel space, as rootkits or event bootkits, hasten security specialists to migrate efforts to this field. One of the first public attacks using virtualization technology to subvert an OS entirely was also a bootkit, the Bluepill[25] attack first presented at Black Hack Conference. The authors of [13] proposed other means of using virtualization to the same goal of Bluepill, using it as a parameter of available public attacks.

The use of a hypervisor, can be made by two fashion: introspection, or within OS's execution. Hypervisors Type 1 or 2 conventionally demands the introspection approach, by acting outside the OS context of execution, usually stopping the Virtual Machine (V.M.) before acting. On the other hand, a hybrid approach is implemented by a kernel or a kernel module, which runs part in the context of the OS and part in the VMM context as well. Being able to act without stopping the V.M., using kernel structures and functions, the hybrid hypervisor reduces the overhead and nullifies the semantic gap [4] present into introspecting the V.M. Therefore, having a context of execution with higher privileges them the OS kernel, is an explored approach for kernel security and advanced threats mitigation, by having an isolated environment through CPU virtualization technology.

[4]Release 1809 in the time of attack.

3.1. Academic hypervisors

Works as Overshadow[2], InkTag[8], SymCall[14], uses a similar approach of hypervisor Type 1 introspection by manipulating the page mapping of physical memory to guarantee security demanded on each specified scenario. The authors of Secvisor[26] proposes a system in which a trust-worthy user's process can verify an untrusted OS behavior and code, with a small degree of assistance from a small and trusted hypervisor.

However, these works mentioned can be categorized as a conceptual approach, as they do not stress its solution against any real-world threat or even a predetermined attack scenario that does not rely on conjectures. More recent works provide a more technical and real-world approach as HyBIS [24], which uses the VirtualBox extension interface to do the introspection for analysis Windows's kernel structures of process. It uses Rekall Forensics [28] alongside with the introspection to dynamic dump portions of the memory without pausing the V.M. With this approach, the authors can detect and stop rogue and hidden user or kernel threat, commonly used by advanced and persistent threats to run and sustain code execution.

In contrasty with HyBIS, the authors of U-HIPE [15] developed a hypervisor for hardware virtualization using Intel Vt-x and an introspection approach to protecting the user and kernel space. This solution mainly aims to monitor guest virtual machine's (V.M.) memory pages, protecting components as process' thread stacks, heaps, and loadable modules to be read or written by threats. The protection is achieved by using Extended Page Table (EPT)[9], which is the Intel implementation of Second Level Address Translation, and providing an Input/Output Memory Management Unit (IOMMU) to the V.M. Using these technologies, U-HIPE is capable of intercept the mapping between the guest-physical, host-physical and direct memory access (DMA) with IOMMU, to hook user and kernel structs or functions.

The author alleged that U-HIPE was stressed against polymorphic/packed malware, hook, and code injection to interfere with process execution. However, they did not test it for any recent –in the time of the work– malware family, exploit or hook/injection techniques. The use of an exploit for Microsoft Office 2007 and the rootkit Zeus from 2009 in the year of 2015 may not correctly show the efficiency of the protection for more recent advanced threats.

3.2. Industry hypervisors

As a result of the rising of advanced threats in the desktop environment, the industry also started to provide solutions based on a hypervisor protection system. In contrast to the academics related works, the industry approach was motivated to protect against up to date real-world threat. The first to hit the market was Kaspersky Secure Hypervisor [12] in 2016, which was designed to work as a Type 2 hypervisor, running at the top of a micro-kernel (KasperskyOS). It works by providing secure domains in the virtualization concept, allowing the communication between the domains following predetermined rules and heuristic. Each domain work as memory isolation for V.M., where the hypervisor arbitrates each access. Although protected by patents and industrial secrecy, its work is very similar to academics works as U-HIPE or a community project as Qubes OS [23]. Qubes OS is from the same author of the Bluepill, and it is an OS that divides tasks into groups according to each hardware isolation domain, using the Xen Project as the base

hypervisor.

After being in closed tests for business partners and government agencies, in 2018, Microsoft delivered a protection system for the public using a hybrid hypervisor within the OS The protection was integrated into the Microsoft Windows Defender Advanced Threat Protection (ATP) [19], working at the top of the Windows's kernel and through nested virtualization technology. This fashion of virtualization allows the security hypervisor to run even if the environment already had another hypervisor, as in a Virtual Machine or by other Windows 10 virtualization technology, for instance, Windows Early Launch AntiMalware [20] or Device Guard [21].

The primary workload of Windows ATP is to monitor the interface between user-level and kernel services, e.g., IOCTL or kernel filters, and sensoring kernel memory and its internal access, i.e., device drivers access to physical memory. This monitoring and sensoring result in reports to the administrative panel, where the user can check if there is any anomalous behavior in-kernel interface or its internals, for further investigation. The Windows ATP Research Team published a report [17], where they demonstrated how an anomalous behavior by a Huawei device driver detected by Windows ATP, led to an investigation which resulted in a privilege escalation flaw published as CVE-2019-524. Although this protection system can detect and report anomalous behavior, according to Microsoft kernel analysis, it cannot stop any attack from happing in the system, besides generating reports which can lead to patching these flaws.

4. Lokke security hypervisor

This work's objective is to design a mechanism to harden the kernel space against any advanced kernel threat's class of attacks, not relying upon any specific vector of attack, but in security policies enforced by the hypervisor. Complementing the afore-mentioned reviewed works, including the industry solutions, the approach of virtualization chosen was the hybrid virtualization approach, where the Virtual Machine Manager VMM (or hypervisor) can run side by side with the OS kernel, virtualizing when needed only the CPU and memory access.

This hybrid approach allows the solution to be partially independent across multiples OS, as the VMM runs in a more privileged execution context than the OS kernel. Therefore any specific OS operations can be made either in OS context of execution (e.g., device driver) or in the VMM virtual CPU handlers, like the ones for V.M. memory or register access. The proposed solution first aimed to work and solve the Windows 10 threat scenario, but the built infrastructure can be integrated into other OS; this possibility will be covered in future steps of the research.

4.1. Technologies involved

Virtualization technologies are dependable on the specific set of instructions in the CPU, even when on the same architecture. There are many hardware virtualization technologies, e.g., Intel Vt-X, AMD-V, ARM-VEx, but the hypervisor core cannot support simultaneously more than one, as a consequence of the instruction set. For this reason, just one technology was chosen, the Intel VT-x [9], mainly because of the higher market share of its CPU, better architecture documentation, and better support for nested hypervisor in industry virtualization solutions. The last criteria mean that the proposed solution can run

in environments already virtualized, where the OS is at the top of Type 1 or 2 hypervisor. Along with Intel's Virtual Machine Extensions (VMX), from Intel Vt-X, the hypervisor will also use the following Intel extensions: EPT, for Second Layer Address Translation (SLAT) and Vt-D, for I/O Memory Management Unit (IOMMU).

The use of EPT is to protect memory pages, by intercepting the mapping between the guest-physical and host-physical addresses resolution with SLAT, imposing restrictions to the protected addresses when needed. This protection may be insufficient, as the authors of the work[32] demonstrated an attack in the Direct Memory Access (DMA) to overlap Intel Vt-D protection for a driver's memory domain. Therefore, IOMMU is needed for similarly intercepting DMA as for physic addresses resolution.

In the early stage of research, one hypervisor was built using the Intel Architectures Software Developer's Manual[9], including its examples, to understand the basics of architecture and CPU extension provided by Intel. This barebone hypervisor was build to mainly be a testbench for interrupt and exception handling, multi-processor support, virtual machine extensions (VMX) instructions, and the Intel Virtualization Technology (V.T.). Although it was technically working as a hypervisor, i.e., being able to create virtual CPUs, it was not compatible with any modern OS

A modern hypervisor needs to simulate or be able to bypass a bunch of hardware events without interfering on its execution or behavior, risking break the OS kernel or even its execution, i.e., handling a hardware interruption without a proper handler. For this reason, building a hypervisor that works and is compatible with any OS is an entirely different workload. Common hypervisor misbehavior with the OS is inherent in how the second registers the Advanced Programmable Interrupt Controller (APIC) for each CPU or the presence of Unified Extensible Firmware Interface (UEFI) in the boot chain, which can break the hypervisor flow to the OS

Therefore, to build a security system based on a hypervisor and not build an utterly new hypervisor from scratch, the project used two open-source and public hypervisors solutions. The first one was SimpleVisor[11], a simple and almost barebone hypervisor for Intel Vt-X, which is compatible with Windows and UEFI. Although it is not tested to boot Windows or any OS, the valuable part of its development for the proposed solution is in its simplicity. The routine to handle the entry of a Virtual Machine and capture the OS context is built in just ten instructions, making the security hypervisor both functional and simple. The other project used as a base to the proposed solution is Bareflank[3], a lightweight hypervisor SDK build and tested to work for Windows or Linux with UEFI. Thus it can be considered lightweight, mainly because general-purpose hypervisor can be so complex as an OS kernel itself; it is built to provide routines and features to other hypervisors as an SDK.

4.2. Hypervisor workflow

The Virtual Machine Extensions (VMX) instructions set provides Intel's CPU with the capabilities to create Virtual Machine and hypervisors, or Virtual Machine Manager. Those instructions, also called VMX operations, are divided into two modes of operations, root and non-root operation. The VMX root operation is used by the hypervisor to create and control the virtualized environment, while the guest software uses the non-root operation. The creation of a virtual environment is done by calling the vmxon instruction, which

enters the logical CPU into a VMX root operation.

From this moment, there will be logical CPUs, inside a VMX root operation, and virtuals CPUs in a non-root VMX operation. Inside a root operation, the hypervisor configures each of the V.M. by filling a structure named Virtual Machine Control Structure (VMCS), this configuration defines the level of virtualization and the behavior from both logical and virtual CPU. Consequently, the hypervisor now controls the logical CPUs, being able to control the hardware and its events, and the virtualized software as the OS In the other side, the virtual CPU is in a non-root VMX operation, have limitations in comparison with a no virtualized environment, i.e., some instructions now pass through the hypervisor, generating a transition named V.M. Exits.

While in a VMX context, there are transitions between the root and non-root VMX operation, named VMX Transitions: to non-root VMX are called V.M. Entries and to VMX root are called V.M. Exits. Those transitions are swap of context between the hypervisor and the virtual machine. The hypervisor can configure handlers for specific V.M. Exits, handle those events according to its need, and then give the execution back to the V.M. with a V.M. Entry. V.M. Exits are significant for the security hypervisor, as they represent events where the hypervisor can execute its heuristic and controls the OS if needed.

4.3. Security Hypervisor

The security hypervisor was tested first as a Windows 10 kernel driver, as the driver runs with kernel's privilege, it starts the virtualized environment doing a VMX operation to create the hypervisor. Although the hypervisor can fully control the CPU when in the root operation, it can brick the OS context of the execution or lead the CPU to an unknown state if it calls any kernel function. That happens for many reasons, as the need for all hypervisor's code is already cached in the CPU or the use of Special Register in kernel calls, which can lead to inconsistencies when doing V.M. Entry. Thus, any work did by the hypervisor must be done exclusively by its functions and CPU instructions. For this reason, the security system is divided into three parts:

- Module infrastructure: contains all the functions to interact with the user and kernel space, and the calls to create and start the hypervisor.
- Common infrastructure: inside the driver context, a set of structures for data to I/O between the kernel and hypervisor.
- Hypervisor infrastructure: all functions to manage the virtual machine and a set of handlers to control its behavior according to the security policies.

The common context includes, for instance, data from functions as *PsLookupProcessByProcessId*, which returns the structure EPROCESS, used to iterate in all processes of the OS The hypervisor handlers can interfere in how the hardware is accessed from the V.M., e.g., access to physical memory, DMA and special registers, as CR3 or CR4. The work to be done by the kernel driver, besides starting the hypervisor, will be limited to gathering information to the hypervisor. This information is used by the hypervisor, avoiding the need to scan memory regions across it. An example of data gathered is the pointer to the *EPROCESS* queue or the list of registered callbacks in the kernel. All the security policies and approaches for control and protect the Windows kernel or the hypervisor itself will do this process.

As stated before, the hypervisor is developed to work more like a security infrastructure, for this reason, the first stage of its infrastructure was dedicated to which V.M. Exit reason it can handle to interfere with Windows execution with the minimum overhead. Therefore, one of the prototypes added handlers to Control Register (C.R.) access.

These register controls or changes the CPU's behavior in the execution context, as the interrupt control, addressing mode switching or paging control. Some registers, as CR1 or CR4, are bitmaps with a series of flags or control information, and in the Intel 64 arch, a new set of registers were added, as the Extended Feature Enable Register (EFER) enabling the *SYSCALL/SYSRET* instruction. Thus, adding handlers for specifics reasons, the V.M. will generate V.MExits to the hypervisor, so it can both do its heuristic and enforcement of policies if any was added.

With that in mind, every time the V.M. executes an action in which a handled C.R. register is accessed, the hypervisor can control its access, and some times, execute a security policy. For example, every time the Windows kernel is switching a process context, it uses the lowest 12 bits of CR3 to store the process-context identifier (PCID), when the PCID Enable bit in CR4 is set. In this scenario, the hypervisor can intercept a process switching before it happens, which can be used for many security policies, e.g., stop a rootkit from starting a rogue process inside the kernel context.

Another VMExit handling implemented was for Extended Page Table (EPT), which adds a second translation phase for the guest's virtual address (GVA) into a physical address, named guest's physical address (GPA). EPT can have access rights for GPA if enabled by the hypervisor, with three different access types: read, write, and execute access. If access is denied for some reason, the corresponding access type on the guest's physical page will trigger a VMExit, which is then handled by the hypervisor according to its security policies. That is used in the protection against attack scenarios researched and described in Section 2, where a user process can access kernel objects directly. Therefore, by providing an infrastructure to regulate and enforce security policies when needed in the hypervisor core, it can be flexible enough to not only protect against a specific vector of attack but protect against the threat inherently in the OS behavior.

5. Real-world threat mitigation

The first stage of this work is concluded and shows promising results by using the hypervisor's infrastructure to mitigate process security token elevation. In Section 2, a repository (UACME) [7] of techniques using Windows's features to process elevation was presented, as its attacks weren't blocked by the works described in Section 3; Windows ATP can detect but can't block any out-of-the-box. Many of UACME attacks change a process's access token [16] to one with elevated permissions, by subsequently testing each attack until one of 62 attack works. To test Lokke's efficiency, Windows 10 release 14393 image was used, so all attacks still unpatched. The first test of the hypervisor's infrastructure was to mitigate process security token elevation, by copying or duplicating an elevated process token access to the attacker process. This attack generally happens in two scenarios: 1) After hijacking an elevated process execution, which copies a privileged access token to the attacker process; 2) Having kernel object manipulation, where the attacker process iterate among system process to copy the access token to the attacker process.

To mitigate both scenarios without the need to know any attack internals, i.e.,

know each possible scenario's particularity, the security policy was to enforce no token elevation. Windows has mechanisms to manage access token, which includes the security permissions. Two Windows API functions make this management[5], where one allows enabling or disabling available privileges in the token, and the other creates a new token with limited privileges based in the one owned by the process. The unique legit way to run a process with elevated privileges is if an already elevated process creates a new process, or if the PE32's manifest contains the need for elevation, which is done by User Account Control (UAC)[16] before the process is launch. Thus, there is no legit way to a process having its access token elevated beside an attack.

With the access token assurance, the hypervisor can control all process token, watching if any privileged process, e.g., system services, does not have its token copied, and monitor if any nonsystem process has its token elevated by any means; which can indicate a Windows security flaw. This security policy is done using the following security system infrastructure: EPROCESS pointer passed from kernel driver to hypervisor, CR3 VMExit handler, and the kernel work queue for thread termination; this queue is used to force a thread being gracefully terminated by the kernel scheduler.

The hypervisor code to control this policy is added inside the VMExit handler of CR3 access. After the first VMExit happens, the following acts take places:

1. It follows the SMSS's EPROCESS field ActiveProcessLinks's ForwardLink pointer, which points to the next process until all process is accessed.
2. For each process, adds its PID into the hash table, indexing a copy of the process token.
3. Any process which has a privileged token, e.g., system process, is added in a special list inside the hypervisor's memory.

After this initialization, the hypervisor checks whenever a process context swaps, by CR3 access, and verify its token with the one copied in the controlled list. If the token was elevated, the process is added to the termination queue. Having this security policy, 50 of 62 attacks in UACME's repository failed, although they worked without the hypervisor protection. The 12 attacks that were still working were not token hijack or elevation, relying on Windows bug, as technique 61 or 57, where an already elevated thread or process executes the command but does not change any access token. Other attacks as 6, 11 or 54 works only in past versions of Windows or exclusively on 32 bits architecture.

Debugging the Windows kernel for any of 50 attacks indicated that the elevated process was terminated before it went into execution, after having the token changed. As the UACME does not use any kernel attacks, another test was made using the discovered HWMonitor's driver flaw[27], to access the physical memory directly and change a dummy process's token to another with system privileged; the dummy process was killed before it went to execution as expected.

6. Conclusion

The results so far demonstrated the security system's potential to enforce security policies seamlessly for the Windows kernel, without knowing the specific vector of attack,

[5] AdjustTokenPrivileges and CreateRestrictedToken

besides the behave that is abused. The approach of change a kernel behave without doing introspection or change in the kernel's code proved to be efficient and seamless. VMWare vSphere Management SDK[31] was used to metric the deviation of the overhead from the virtualized Windows kernel, and it when nested virtualized with Lokke's hypervisor, resulting in a lower than 1% deviation for both memory and CPU use.

It is expected that in the next steps, with full support for Second Layer Translation (SLAT) and IOMMU, the hypervisor been mature enough to be tested against sophisticated rootkits, which tries to hide inside the kernel or uses the own hypervisor to avoid detection. Although not fully implemented yet, the Intel PTE extension handlers can intercept physical memory access through SLAT and avoided tree critical Windows 10 exploits from working, including the HWMonitor's exploit. These attacks were blocked by the policy of memory access being made from userspace directly to kernelspace. They are:

- Asus Memory Mapping Driver (ASMMAP) [30]: able to map or unmap to attacker process the physical memory device, with R/W permissions.
- Intel Network Adapter Diagnostic Driver (iQVW64)[29]: able to do a large set of operations, such as R/W kernel virtual memory or allocate/deallocate memory pool.

This workload is similar to how Supervisor Mode Access Prevention (SMAP) works on Intel Broadwell and newer microarchitecture; its intrinsic work can be found starting at page 408 of Intel's manual[9]. SMAP is not enabled by default in many Windows 10 desktops, and even if enable, Windows does not monitor the correspondent CR4 bitmask changing, allowing kernel exploits to bypass this control. The use of the Intel PTE to control memory access, and not only guarantee CR4's integrity to SMAP, was made to test the PTE handlers and workload. In the future, SMAP and SMEP (Supervisor Mode Execution Prevention) will be enforced by trivially handling CR4 access, in combination to support to kernel memory leak and direct kernel object manipulation protection.

References

[1] ATTACK, K. A. T. Advanced threat defense and targeted attack risk mitigation. Tech. rep., Kaspersky, 2019.

[2] CHEN, X., GARFINKEL, T., LEWIS, E. C., SUBRAHMANYAM, P., WALDSPURGER, C. A., BONEH, D., DWOSKIN, J., AND PORTS, D. R. Overshadow: A virtualization-based approach to retrofitting protection in commodity operating systems. *SIGARCH Comput. Archit. News 36*, 1 (Mar. 2008), 2–13.

[3] DEVELOPMENT TEAM, B. lightweight hypervisor sdk written in c++ with support for windows, linux and uefi. `https://github.com/Bareflank/hypervisor`.

[4] DOLAN-GAVITT, B., LEEK, T., ZHIVICH, M., GIFFIN, J., AND LEE, W. Virtuoso: Narrowing the semantic gap in virtual machine introspection. In *2011 IEEE Symposium on Security and Privacy* (May 2011), pp. 297–312.

[5] DONG, Y., ZHANG, X., DAI, J., AND GUAN, H. Hyvi: A hybrid virtualization solution balancing performance and manageability. *IEEE Transactions on Parallel and Distributed Systems 25*, 9 (Sep. 2014), 2332–2341.

[6] GRAZIANO, M., FLORE, L., LANZI, A., AND BALZAROTTI, D. Subverting operating system properties through evolutionary dkom attacks. In *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2016).

[7] HFIREF0X. Defeating Windows User Account Control by abusing built-in Windows AutoElevate backdoor. `https://github.com/hfiref0x/UACME`, 2018.

[8] HOFMANN, O. S., KIM, S., DUNN, A. M., LEE, M. Z., AND WITCHEL, E. Inktag: Secure applications on an untrusted operating system. *SIGPLAN Not. 48*, 4 (Mar. 2013), 265–278.

[9] INTEL. *Intel 64 and IA-32 Architectures Software Developer's Manual - Volume 3C*. Intel Corporation, Setember 2016.

[10] INTELLIGENCE, W. T. Microsoft security intelligence report. Tech. Rep. 24, Microsoft, 2019.

[11] IONESCU, A. Simplevisor is a simple, portable, intel vt-x hypervisor. https://github.com/ionescu007/SimpleVisor.

[12] KASPERSKY. Kaspersky secure hypervisor. https://os.kaspersky.com/products/kaspersky-secure-hypervisor, 2016.

[13] KING, S. T., AND CHEN, P. M. Subvirt: implementing malware with virtual machines. In *2006 IEEE Symposium on Security and Privacy (S P'06)* (2006), pp. 14 pp.–327.

[14] LANGE, J., AND DINDA, P. Symcall: Symbiotic virtualization through vmm-to-guest upcalls. vol. 46, pp. 193–204.

[15] LUȚAȘ, A., COLEȘA, A., LUKACS, S., AND LUTAS, D. U-hipe: hypervisor-based protection of user-mode processes in windows. *Journal of Computer Virology and Hacking Techniques* (02 2015).

[16] MICROSOFT. Access tokens. https://docs.microsoft.com/en-us/windows/desktop/secauthz/access-tokens.

[17] MICROSOFT. From alert to driver vulnerability: Microsoft defender atp investigation unearths privilege escalation flaw. https://www.microsoft.com/security/blog/2019/03/25/from-alert-to-driver-vulnerability-microsoft-defender-atp-investigation-unearths-privilege-escalation-flaw.

[18] MICROSOFT. Securing device objects. https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/securing-device-objects.

[19] MICROSOFT. Windows defender advanced threat protection. https://www.microsoft.com/en-us/windowsforbusiness/windows-atp.

[20] MICROSOFT. Windows early launch antimalware. https://docs.microsoft.com/en-us/windows-hardware/drivers/install/early-launch-antimalware, 2017.

[21] MICROSOFT. Windows defender application control configurable code integrity and virtualization-based security. https://docs.microsoft.com/en-us/windows/security/threat-protection/device-guard, 2018.

[22] ORMAN, H. The morris worm: A fifteen-year perspective. *Security and Privacy, IEEE 1* (10 2003), 35 – 43.

[23] OS, Q. Qubes os, a reasonably secure operating system. https://www.qubes-os.org.

[24] PIETRO, R. D., FRANZONI, F., AND LOMBARDI, F. Hybis: Windows guest protection through advanced memory introspection. *CoRR abs/1601.05851* (2016).

[25] RUTKOWSKA, J. Subverting vista kernel forfun and profit. *Black Hat Briefings* (7 2006).

[26] SESHADRI, A., LUK, M., QU, N., AND PERRIG, A. Secvisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity oses. In *SOSP* (2007), T. C. Bressoud and M. F. Kaashoek, Eds., ACM, pp. 335–350.

[27] SILVA, O. A. A. NIST CVE-2018-8061. https://nvd.nist.gov/vuln/detail/CVE-2018-8061, 2018.

[28] SYSTEMS, V. Rekall forensic. http://www.rekall-forensic.com/.

[29] UNKNOW. Intel Network Adapter Diagnostic Driver IOCTL Handling Vulnerability. https://www.exploit-db.com/exploits/36392, 2015.

[30] UNKNOW. ASUS Memory Mapping Driver - Physical Memory Read/Write. https://www.exploit-db.com/exploits/39785/, 2016.

[31] VMWARE, I. vSphere Management SDK v7. https://code.vmware.com/web/sdk/7.0/vsphere-management, 2020.

[32] WOJTCZUK, R., AND RUTKOWSKA, J. Following the white rabbit: Software attacks against intel (r) vt-d technology.

[33] YOSIFOVICH, P., RUSSINOVICH, M. E., SOLOMON, D. A., AND IONESCU, A. *Windows Internals, Part 1: System Architecture, Processes, Threads, Memory Management, and More (7th Edition)*, 7th ed. Microsoft Press, Redmond, WA, USA, 2017.