International Journal of Advanced Research in Engineering Technology & Science

ISSN: 2349-2819

Email: editor@ijarets.org Volume-9, Issue-8 August– 2022 www.ijarets.org

Security of the Virtual Machine using Secure Hardware Enforced Hypervisor

Anshu Mali Bhushan
Research Scholar
Dept. of Computer Science
Maharaja Agrasen Himalayan Garhwal University, Uttarakhand

Dr. Sanjoli Kaushik
Assistant Professor
Dept. of Computer Science
Maharaja Agrasen Himalayan Garhwal University, Uttarakhand

ABSTRACT

With the advent of Virtualization came the threat of attacks due to Malware such as APT (Advanced Persistent Threats) and Kernel Mode threats. These attacks are mostly Zero-day type vulnerability for the host operating system and hence nearly impossible to detect and hence lead to unfathomable loss of time and money!! Sometimes it can push a whole country's projects back by several years.

The criticality of these attacks is further enhanced as their attack is not on the user space but on kernel space which directly control the hardware such as microprocessor or registers which store highly critical information such as registers etc. Thus, this Hardware must be protected at all costs to avoid system compromise.

We are now introducing Hardware Enforced Security (HES) into our research which shall be looking after ensuring information state protection in the hardware registers and microprocessors or micro controllers (IoT). Lots of manufacturers have implement HES into their design but OS developers are not leveraging this information to its true potential. Hence, we have now developed a complete package using HES extensions to ensure protection to critical areas.

Our approach shall describe in detail this method and will also demonstrate an experiment which has confirmed that HES can indeed protect APT / Kernel Mode Attacks.

INTRODUCTION

Virtualization addresses IT's most pressing challenge: the infrastructure sprawl that compels IT departments to channel 70 percent of their budget into maintenance, leaving scant resources for business-building innovation.

The difficulty stems from the architecture of today's x86 computers: they're designed to run just one operating system (OS) and application at a time. As a result, even small data centers must deploy many servers, each operating at just 5–15 percent of capacity—highly inefficient by any standard.

Virtualization solves the problem by enabling several OS and applications to run on one physical server or "host." Each self-contained "virtual machine," which comprises a guest OS and an application, is isolated from the others. It uses as much of the host's computing resources as it requires without affecting other virtual machines running on the same host.

Secure Hypervisor has been developed from the ground up to run virtual machines in a secure manner and incorporates many powerful security features that address the security concerns of the most demanding data center environments for enterprises and government organizations. The holistic security architecture of **Secure Hypervisor** achieves this goal by providing security mechanisms at multiple layers.

- Secure isolation of virtual machines at the virtualization layer. This includes secure instruction isolation, memory isolation, device isolation, and managed resource usage and network isolation.
- Configurable secure management of the virtualized environment. This includes secure communication between virtualization components via SSL; host protection via lockdown mode; and least privilege by a fine-grained, role-based access-control mechanism.
- Secure deployment of the software package on servers through use of various platform-integrity mechanisms such as digitally signed software packages and Intel Trusted Platform Module (TPM)–based trusted boot.
- Rigorous secure software development life cycle that enables developers to create software using secure design and coding principles such as minimum attack surface, least privilege, and defense in depth.

Recent years have been memorable for cybercrimes against high-profile targets, it included major attacks on defense establishments, corporates, social networking sites, movie-streaming giants, music services etc. Conventional security mechanisms are not effective against the advanced form of attacks that are frequently employed against these high value targets.

A standout feature of many of these attacks is the complexity of AI embedded malware making conventional security solutions woefully inadequate to detect or prevent them.

Threats that avoid detection and harvest valuable information over a long time are known as Advanced Persistent Threats (APTs). Traditional security measures such as antivirus, firewalls etc. cannot provide protection against APTs thereby leaving systems vulnerable to data breaches. The consequence of an APT attack is devastating as the attack may continue for a long time uninterrupted due to the limitations of most of the current security solutions.

Hardware Enforced Security (HES) refers to protecting a system against malware attacks by leveraging security features provided by the hardware implemented using a hypervisor.

Secure Virtual Machine Isolation in Virtualization

Virtualization as it is done today has changed considerably since 2006. Then, it was done primarily in software using binary translation, a method that modifies sensitive instructions on the fly to "virtualizable" instructions. With advances in CPU technology, virtualization capabilities are built directly into the CPU. Today, the hypervisor primarily is the management interface to the hardware primitives. Isolation of CPU, memory, and I/O now is done at a hardware level, with the hypervisor managing how much of the hardware resources a virtual machine can use, similar to a choreographer or traffic officer. This part of the hypervisor is called the virtual machine monitor (VMM). With the ability to leverage these CPU extensions, the attack surface of the hypervisor shrinks considerably. Many security-related concerns about virtualization are unwarranted. Multiple hardware- and softwaresupported isolation techniques—

as well as other robust security mechanisms such as access control and resource provisioning—address the risks associated with these worries.

Instruction Isolation

From a security standpoint, a primary concern is that of a virtual machine's running in a highly privileged mode that enables it to compromise another virtual machine or the virtual machine monitor (VMM) itself. However, Intel VT-x and AMD-V extensions don't enable virtual machines to run at "Ring-0."(Yujuan Jiang,Xiangyang Li,Binlai An; 2020) [1]

Only the virtual machine monitor (VMM) runs at a hardware privilege level; guest OSs run at a virtualized privilege level. The guest OS does not detect that it is running at a nonprivileged virtualized level. As previously mentioned, when the guest OS executes a privileged instruction, it is trapped and emulated by the virtual machine monitor.

Intel Hyper-Threading Technology (Intel HT Technology) enables two process threads to execute on the same CPU core. These threads can share the memory cache on the processor. (Shi-Wu Lo,Kam-YiuLam,Wen-Yan Huang,Sheng-Feng Qiu (2013) [2]

Security Hypervisor do not provide Intel HT Technology to the guest OS. Security Hypervisor, however, can utilize it to run two different virtual machines simultaneously on the same physical core if configured to do so.

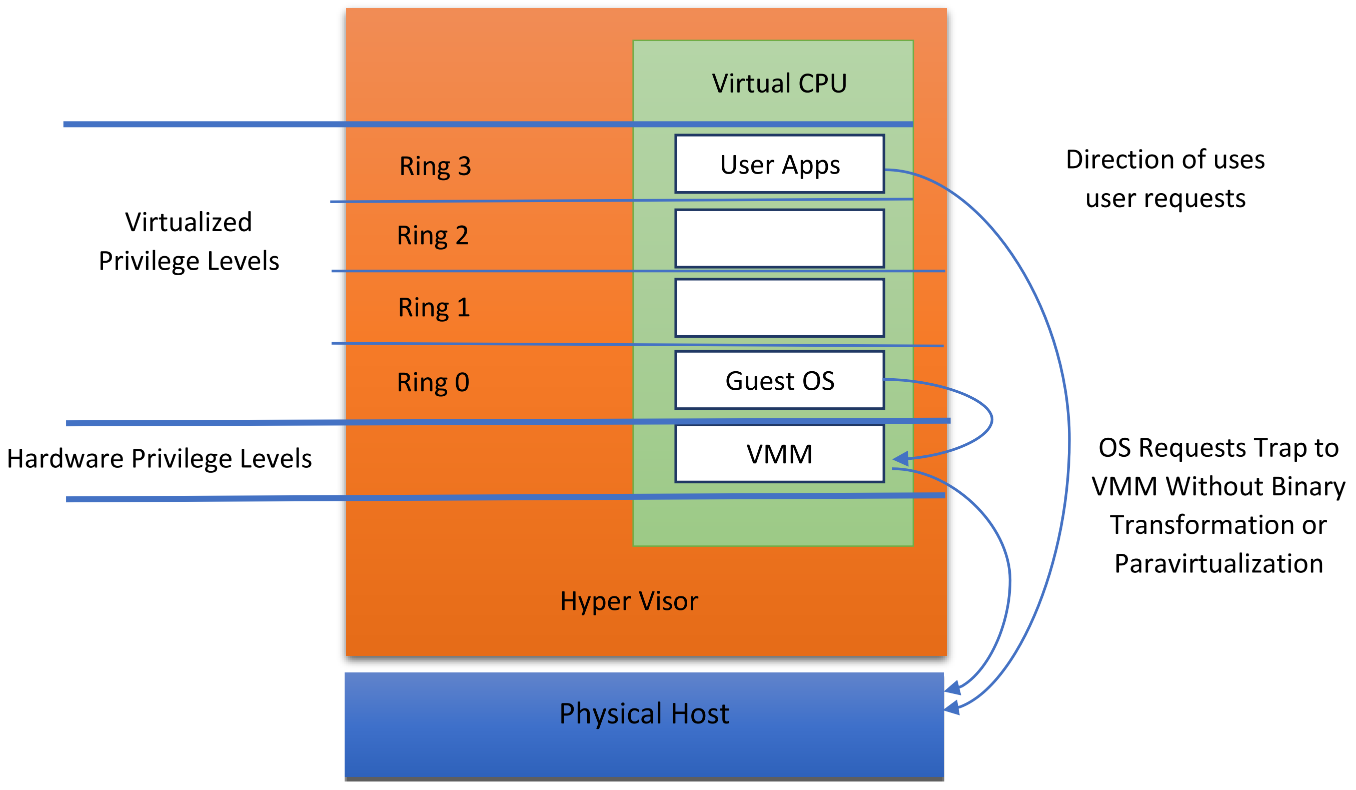
Virtual CPU
Ring 3
User Apps
Ring 2
Ring 1
Ring 0
Guest OS
VMM
Virtualized Privilege Levels
Hardware Privilege Levels
Direction of uses user requests
OS Requests Trap to VMM Without Binary Transformation or Paravirtualization
Hyper Visor
Physical Host

Fig 1: Instruction Isolation

Memory Isolation

The system administrator defines the RAM allocated to a virtual machine by the virtual machine monitor (VMM) via the virtual machine's settings. The virtual machine kernel allocates memory when it defines the resources to be used by the virtual machine. (Eunbyung Park,Bernhard Egger,Jaejin Lee, 2011) [3]. A guest OS uses physical memory allocated to it by the virtual machine kernel and defined in the virtual machine's configuration file. An OS booting on real hardware is given a zero-based physical address space; an OS executing on virtual hardware is given a zero-based address space. The virtual machine monitor (VMM) gives each virtual machine the illusion that it is using such an address space, virtualizing physical memory by adding an extra level of address translation. A machine address refers to actual hardware memory; a physical address is a software abstraction used to provide the illusion of hardware memory to a virtual machine.(Violeta Medina,Juan Manuel García, 2014) [4]

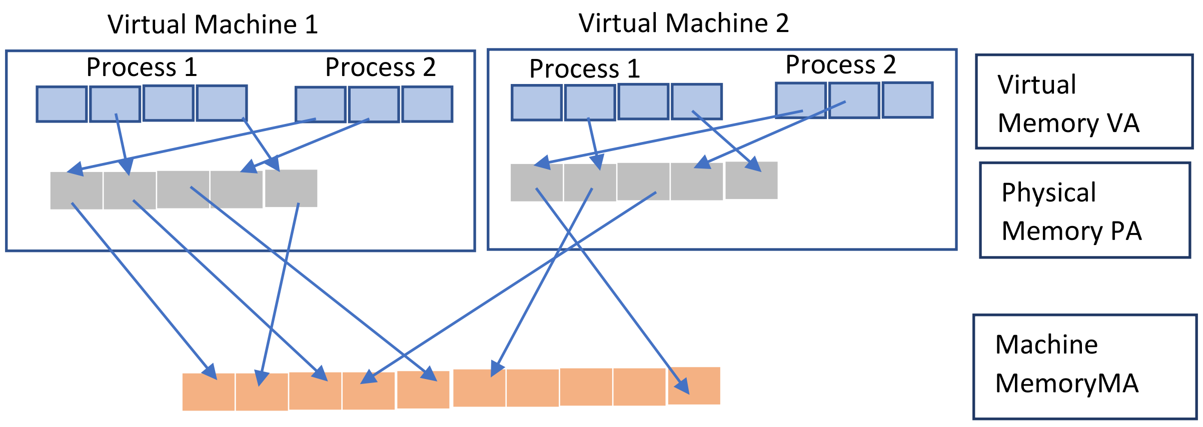

Fig 2: Memory Virtualization

The virtual machine monitor (VMM) maintains a pmap data structure for each virtual machine to translate “physical” page numbers (PPNs) to machine page numbers (MPNs). Virtual machine instructions that manipulate guest OS page tables or translation look-aside buffer contents are intercepted, preventing updates to the hardware memory management unit. Separate shadow page tables, which contain virtual-to-machine page mappings, are maintained for use by the processor and are kept consistent with the physical-to-machine mappings in the pmap. (Carl A. Waldspurger, 2002) [5]

This approach enables ordinary memory references to execute without additional overhead, because the hardware translation look-aside buffer caches direct virtual-to-machine address translations read from the shadow page table. The extra level of indirection in the memory system is extremely powerful. The server can remap a “physical” page by changing its PPN-to-MPN mapping in a manner that is completely transparent to the virtual machine. It also enables the virtual machine monitor (VMM) to interpose on guest memory accesses. Any attempt by the OS or any application running inside a virtual machine to address memory outside of what has been allocated by the virtual machine monitor (VMM) causes a fault to be delivered to the guest OS, typically resulting in an immediate system crash, panic, or halt in the virtual machine, depending on the OS. When a virtual machine requires memory, the virtual machine kernel zeros each memory page out before being handed to the virtual machine. Normally, the virtual machine then has exclusive use of the memory page, and no other virtual machine can touch it or even detect it. The exception is when transparent page sharing is in effect. Transparent page sharing is a technique for using memory resources more efficiently. Security Hypervisor scans the content of guest physical memory for sharing opportunities. Instead of comparing each byte of a candidate guest physical page to other pages, an action that is prohibitively expensive, Security Hypervisoruses hashing to identify potentially identical pages. Memory pages that are identical in two or more virtual machines are stored once in the host system's RAM, and each of the virtual machines has read-only access. (J.J. Donovan,H.D. Jacqby, 1977) [6] Such shared pages are common, for example, if many virtual machines on the same host run the same OS. As soon as any one virtual machine attempts to modify a shared page, it gets its own private copy. Because shared memory pages are marked copy-on-write, it is impossible for one virtual machine to leak private information to another through this mechanism. Transparent page sharing is controlled by the virtual machine kernel and virtual machine monitor (VMM) and cannot be compromised by virtual machines. It can also be disabled on a per-host or per–virtual machine basis.

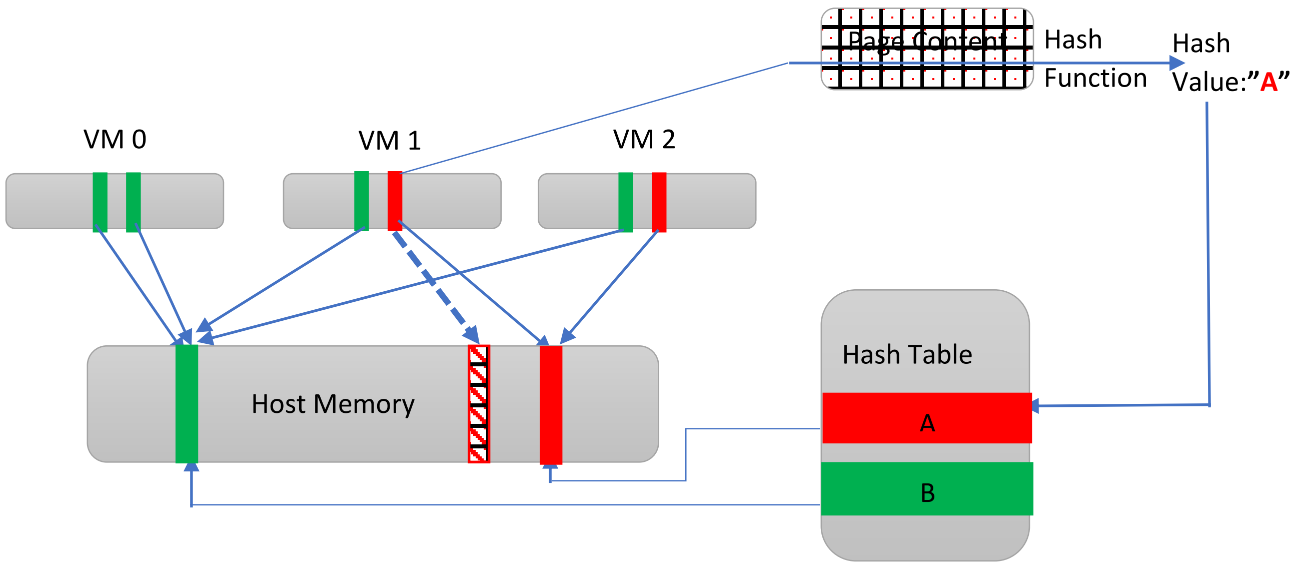

Fig 3: Transparent Page Sharing – Page-Content Hashing

Memory Protection

To protect privileged components, such as the virtual machine monitor (VMM) and virtual machine kernel, Security Hypervisoruses certain well-known techniques. Address space layout randomization (ASLR) randomizes where core kernel modules are loaded into memory. (Mihai Bucicoiu,Lucas Davi,Razvan Deaconescu,Ahmad-Reza Sadeghi, 2015)[7]

The NX/XD CPU features enable thevirtual machine kernel to mark writeable areas of memory as nonexecutable. Both methods protect the system from buffer overflow attacks in running code. (KrerkPiromsopa,Richard J. Enbody, 2006) NX/XD CPU features also are exposed to guest virtual machines by default.

Device Isolation

Each virtual machine is isolated from other virtual machines running on the same hardware. Virtual machines share physical resources such as CPU, memory, and I/O devices; a guest OS in an individual virtual machine cannot detect any device other than the virtual devices made available to it. (Othmen Braham,Guy Pujolle, 2011) [9]

To further clarify, a virtual machine can detect only the virtual (or physical) devices assigned to it by the systems administrator, such as the following examples:

- A virtual SCSI disk mapped to a file on a disk
- An actual disk or LUN connected to a physical host or array
- A virtual network controller connected to a virtual switch
- An actual network controller connected to a physical network

A virtual machine cannot map to a device that has not been preassigned. A virtual machine is incapable of mounting another virtual machine's disk unless the disk has been explicitly assigned to both virtual machines in the Security Hypervisormanagement console.

Device Access to Hardware

At the hardware level, all direct memory access (DMA) transfers and device-generated interrupts are virtualized and isolated from other virtual machines. This prevents one virtual machine from accessing the memory space controlled by another virtual machine. If such an attempt is made by a virtual machine, the guest OS will receive a fault from the CPU. Because the virtual machinekernel and virtual machine monitor (VMM) mediate access to the physical resources, and all physical hardware access takes place through the virtual machinekernel, virtual machines cannot circumvent this level of isolation.

I/O Remapping

Modern processors feature an I/O memory management unit that remaps I/O DMA transfers and device interrupts. This enables virtual machines to have direct access to hardware I/O devices, such as network cards, storage controllers (HBAs), and GPUs. In AMD processors, this feature is called AMD I/O Virtualization (AMD-Vi) or I/O memory management unit (IOMMU); in Intel processors, the feature is called Intel Virtualization Technology for Directed I/O (VT-d). Within Security Hypervisor, use of this capability is called DirectPath I/O. DirectPath I/O does not impact the security properties in any way. For example, a virtual machine configured to use VT-d or AMD-Vi to directly access a device cannot influence or access the I/O of another virtual machine.

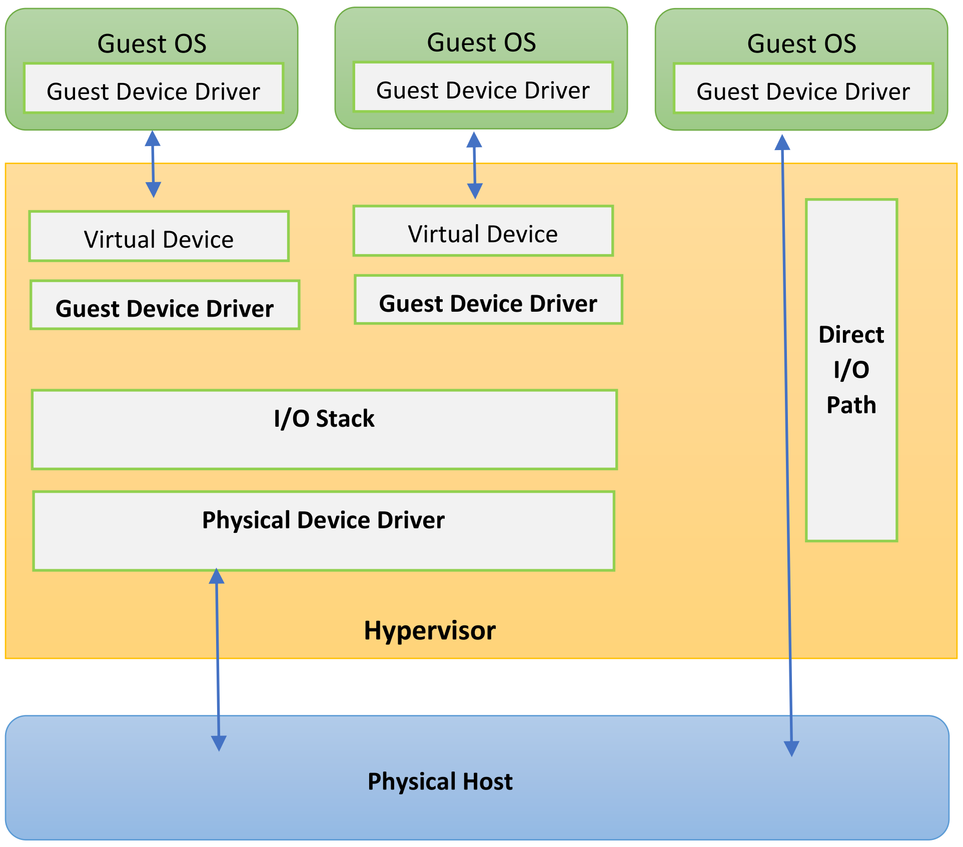

Fig 4: I/O Data Path via the Hyper Visor and Direct I/O Path

Resource Provisioning, Shares, and Limits

In a virtualized environment, resources are shared among all virtual machines. But because system resources can be managed, it enables use limits on virtual machines. There are a number of methods to address this.

Provisioning

In a physical system, the OS can use all the hardware resources. If the system has 128GB of memory, and the OS can address it, all of that memory can be used. The same applies to CPU resources. However, as previously noted, all resources are shared in a virtual environment. An OS using too many resources, CPU for example, potentially can deprive another OS of the resources it needs. Provisioning is the first step in managing virtual machine resources.(Qizhi Zhang,Haopeng Chen,Yuxi Shen,Sixiang Ma,Heng Lu, 2016) [10]

A virtual machine should be provisioned with only the resources it requires to do a job. Because virtual machines never can use more CPU or memory resources than provisioned, users can limit the impact on other virtual machines.

Shares

Users can further isolate and protect neighboring virtual machines from “noisy neighbors” through the use of shares. Grouping “like” virtual machines into resource pools, and leaving shares set to default, ensures that all virtual machines in the pool receive approximately the same resource priority. A “noisy neighbor” will not be able to use more than any other virtual machine in the pool.

Limits

Previous recommendations suggested the use of limits to control resource usage. However, based on more operational experience, it has been found that virtual machine–level limits can have detrimental operational effects if used improperly.

For example, a virtual machine is provisioned with 4GB and the limit is set to 4GB. Then a change is made to increase the memory to 8GB, but the limit is left unchanged. At this stage, when the virtual machine has used up the 4GB and moves to an “extra 4GB,” the 4GB limit causes the virtual machine to experience memory pressure and ballooning or swapping, thereby affecting its operational efficiency and the entire environment. Setting limits on a per–virtual machine basis also can have an impact on operational management and exposes the environment to misconfigurations. Limits are powerful but should be used only when the impact is fully understood.

Secure Management

Any IT infrastructure software must have a means for management, configuration, monitoring, and troubleshooting. Security Hypervisor has been designed to provide a rich set of management features that do not compromise the ability to maintain the security of the platform. First, we will discuss the design of the Security Hypervisorhypervisor; then we will discuss capabilities related to access controls—such as authentication, authorization, and logging— as well as secure communications.

Architecture of Layered Secure Hypervisor

Secure Hypervisor is based on Hardware Enforced Security (HES)which protects the systems against kernel mode threats originating externally as well as internally (insider attacks). In addition to kernel protection, this solution also guarantees hardware enforced application white-listing as well as hardware enforced application screening for detecting malware infections. It a novel way of distributing security functionalities (protection & detection) across kernel module and hypervisor, thereby reducing the performance cost incurred due to frequent traps to hypervisor.

Performance of Secure Hypervisor

Secure Hypervisor does not suffer from performance related concerns compared to other hypervisor-based protection techniques due to the optimized split of security features between the hypervisor and Operating System kernel. (Takeshi Okamoto, 2017) [11]

It describes three important components.

- Secured application
- Kernel security module
- Security hypervisor

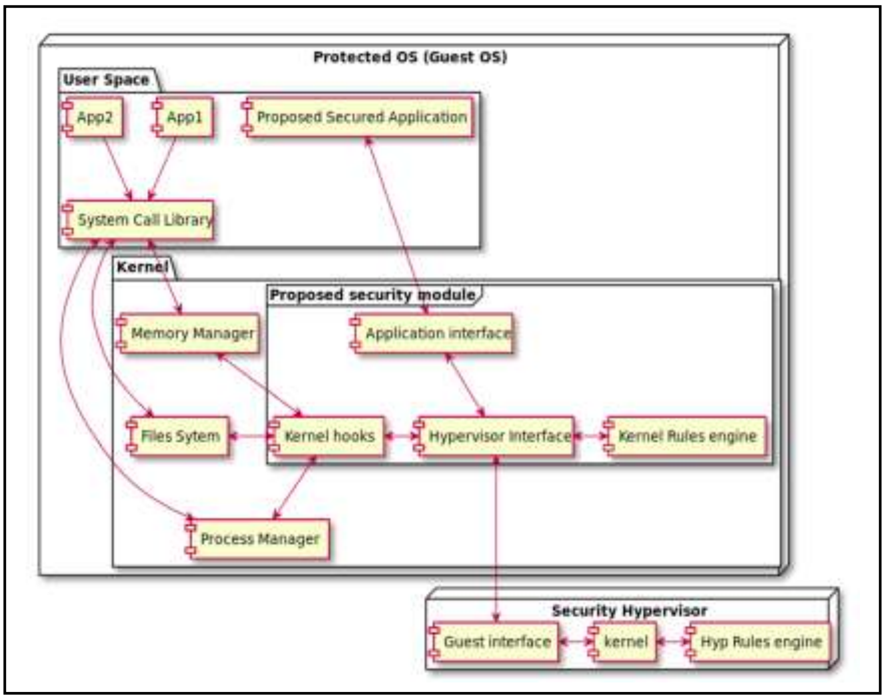

Fig 5: Architecture

Secured Application

Software application which runs in the user-space. This application shall be always running and is responsible for displaying alerts generated by the underlying modules. The application is authorized by security hypervisor by highly secured white-listing method and have a secure channel to communicate with the proposed kernel security module.

Other responsibilities of the secured application would be configurations management and secure updates. (Mahaveerakannan R,Suresh Gnana Dhas C, 2019) [12]

Configuration management shall deal with how the alerts shall be displayed, what actions to be taken for each alert like just logging or killing the application after logging, notifying to registered users etc. Shall have an option to save a log of alerts with time stamp. Secure update involves downloading the update package from authorized server, verify the checksum, decryption of downloaded package and proper installation of the package.

Kernel security module:

This is the software program which runs in the OS kernel privilege mode. Module can be made either part of the kernel by statically compiling into the kernel or can be dynamically loaded after verifying the signatures by the hypervisor. As discussed earlier most of the OSes in industry are monolithic kernels, hence being part of the kernel, module can access the kernel data structures of various kernel sub systems

This module is further split into 4 sub-modules

i. **Kernel Hooks:**This sub-module registers entry and exit functions with critical kernel functions, if OS provides such an interface. In case of no OS interfaces, critical functions are replaced with a wrapper function, which then calls into these OS functions.

Before calling the actual kernel critical functions, the parameters passed and process information (task structure) are passed to the rules sub-module. Depending upon the return status of the rules engine one of the following actions are taken.

1. If return status is false, no hypervisor trap is performed. Regular OS function call proceeds
2. If return status is true, depending upon the call, parameters are prepared (as a secured packet) and a trap to hypervisor is called through hypervisor interface sub-module

Depending upon the return status of the hypervisor call, either regular OS function call proceeds or exit to user-space with permission denied error is called

Once regular OS function execution finishes, exit hook is called, inside which rules sub-modules reference data via hypervisor interface is updated based on the success/failure of OS call

ii. **Rule Engine:**This sub-module can access reference critical kernel data structures pertaining to each process running on the OS maintained in hypervisor protected memory. **Fig 6** represents the data structure stored in the hypervisor memory. The per process data is updated into this data structures whenever a permitted system call succeeds. These updates happen in exit hook of the proposed kernel security module.

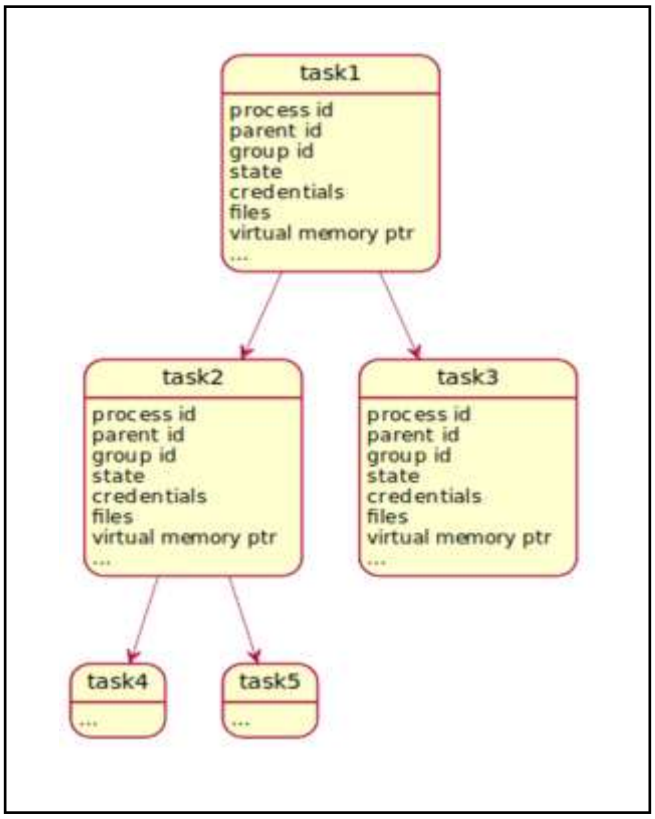

Fig 6: Reference data stored in hypervisor area accessible for rules engine

Logic inside compares the process's credentials with the reference data in hypervisor memory and if a mismatch is found, hypervisor call is made and necessary alert is generated through security hypervisor.

Rules engine also calls hypervisor for secured operations like signature verification of a module/process etc.

Hypervisor interface: This module establishes a secured channel between hypervisor and kernel security module. Secure communication channel details are discussed in a different patent. This interface is used by all other sub modules to communicate with the security hypervisor

Application Interface*:* As mentioned above, this sub-module provides interface to secured application to fetch the required data like alerts and logs generated by itself and the security hypervisor software component. This interface talks only with the authorized application approved by security hypervisor.

iii. **Security Hypervisor:**This is a software image which runs at the highest privilege level. Proposed security hypervisor leverages Hardware Enforced Security features available on the processors and classified as a type 1 hypervisor or bare metal hypervisor. Security hypervisor is the first piece of software which boot-loader loads into memory and startsexecuting. (Geoffrey Papaux,Daniel Gachet,Wolfram Luithardt, 2014)[13]

Major functionalities of security hypervisor are listed below

1. Security hypervisor claims all the memory available, along with the devices
2. It configures a virtual machine environment to have a restricted memory/device access for the OS to be protected.
3. Hypervisor emulates some of the instructions as well as devices access and provide transparent interfaces to enable the guest OS to run unmodified
4. To ensure security or authorized access to some of the resources, hypervisor configures traps whenever some critical processor registers are modified or instructions are executed or memory locations are accessed

Proposed security hypervisor contains primarily three modules

Guest Interface: *As discussed in previous module description,* this module is the other end of secure channel for communication between kernel security module and security hypervisor. Secure communication channel details are discussed in a different research paper to be published.

Kernel:This module has memory initialization, instruction & device emulation code. It has the logic to create necessary virtual machine environment for guest OS to boot.

Hypervisor rules engine:This module contains the reference critical data structures of OS in a restricted memory region, with read-only access to kernel security module. These data structures are modified by rules engine on behalf of the kernel module, passed through the secure communication channel.

It has all the trap functionalities registered. These trap functions get called on access to critical processor registers or memory areas. In these trap functions, decision is taken whether an operation is permitted or not, if such access is unauthorized, necessary alert is raised along with nullifying the action.

Please refer to the **Fig 7** for detailed flow of a calls involved in any system call in a typical monolithic operating system.

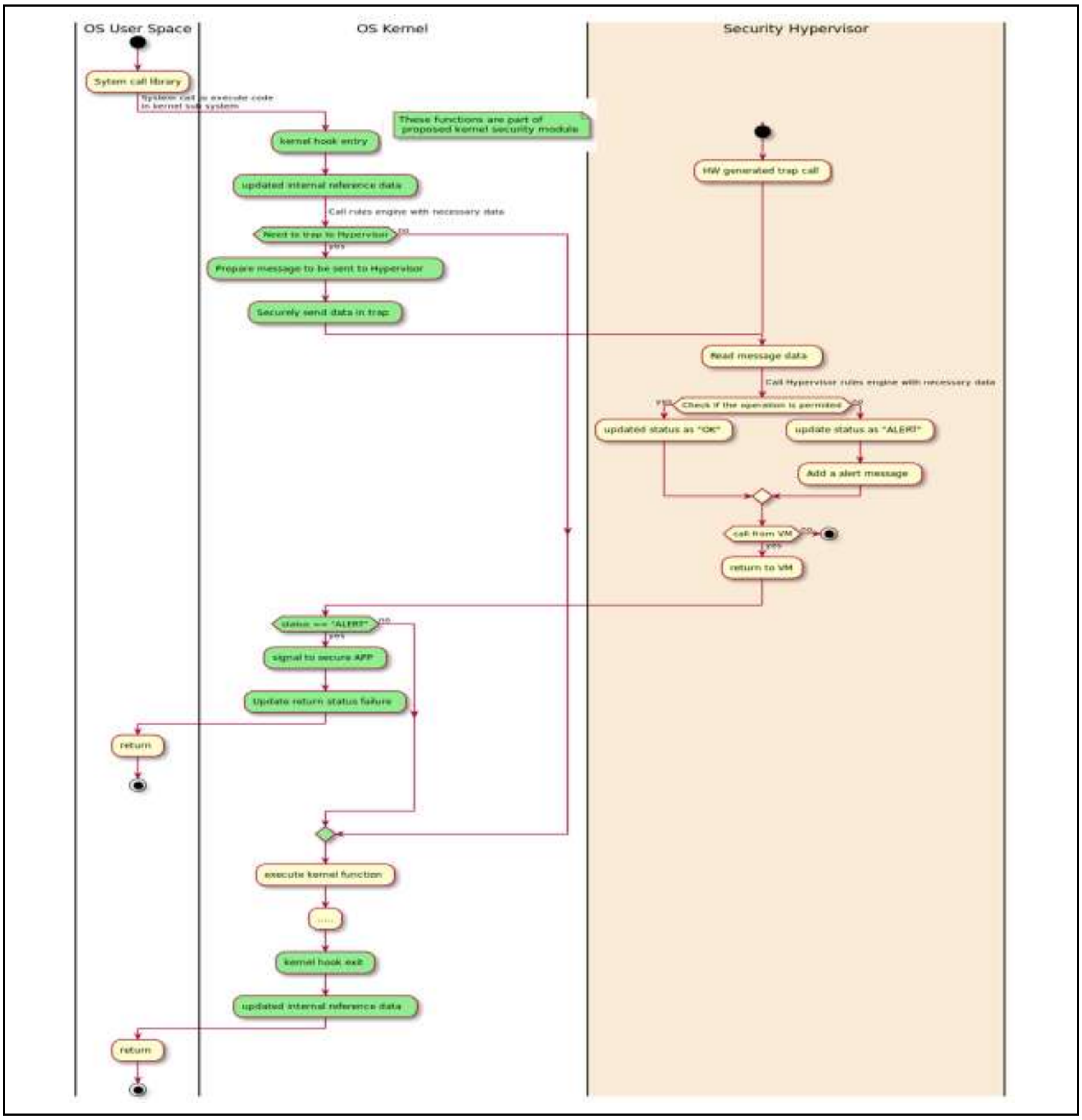
OS User Space
OS Kernel
Security Hypervisor
Sytem call library
System call to execute code in kernel sub system
These functions are part of proposed kernel security module
kernel hook entry
updated internal reference data
Call rules engine with necessary data
Need to trap to Hypervisor
no
yes
Prepare message to be sent to Hypervisor
Securely send data in trap
HW generated trap call
Read message data
Call Hypervisor rules engine with necessary data
Check if the operation is permitted
yes
no
updated status as "OK"
update status as "ALERT"
Add a alert message
call from VM
no
yes
return to VM
status == "ALERT"
no
yes
signal to secure APP
Update return status failure
return
execute kernel function
.....
kernel hook exit
updated internal reference data
return

Fig 7: Detail flow of call involved in System call

Conclusion

1. ***Protection against zero-day vulnerabilities***: Achieved by restricting unauthorized access to critical data structures and processor registers

2. ***Protection against root-kits***: As the protections are distributed across kernel & hypervisor, which has highest privilege level, root-kits can be overpowered

3. ***Performance cost incurred due to frequent traps into hypervisor***: This limitation is overcome by handling protections both in kernel module as well as hypervisor, taking judicious decision on cost of trap versus vulnerability introduced when implemented in kernel module

4. ***Supporting multiple OSes as well as different versions of a OS***: This can be achieved by making all OS/version specific changes in kernel module, which can be different for each OS/version, keeping the security hypervisor changes limited and less prone to vulnerabilities

Keywords

Virtualization; Hypervisor; Virtual Machine; Virtual Network; Secure Configuration; Security

Monitoring; Guest OS

References

[1] Ashish Saxena, Rajeev Saxena, Yogadhar Pandey, An Exhaustive Analysis On Various Foggy Image Enhancement Techniques, International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE) Volume 3, Issue 1, January 2014

[2] Abhishek Tiwari, Pramil Singh, Analysis of Color Contrast Enhancement Techniques, International Journal of Emerging Technology and Advanced Engineering Website:2008 Certified Journal, Volume 4, Issue 4, April 2014

[3] Wanhyun Cho, SeongchaeSeo, Jinho You, Soonja Kang, Enhancement Technique of Image Contrast using New Histogram Transformation, Journal of Computer and Communications, 2014, 2, 52-56

[4] Raju.A, Dwarakish. G. S, D. Venkat Reddy, A Comparative Analysis of Histogram Equalization based Techniques for Contrast Enhancement and Brightness Preserving, International Journal of Signal Processing, Image Processing and Pattern Recognition Vol.6, No.5 (2013)

[5] Suneetha, Dr.T. Venkateswarlu, Enhancement Techniques for Gray scale Images in Spatial Domain, International Journal of Emerging Technology and Advanced Engineering Website: www.ijetae.com (ISSN 2250-2459, Volume 2, Issue 4, April 2012)

[5] S. SomorjeetSingh,Th. Tangkeshwar Singh, H. Mamata Devi, TejmaniSinam, Local Contrast Enhancement using Local Standard Deviation, International Journal of Computer Applications (0975 – 888)Volume 47– No.15, June 2012

ShyamLal, Mahesh Chandra, Efficient Algorithm for Contrast Enhancement of Natural Images, Received April 21, 2012; Accepted August 10, 2012

[6] Sapana S. Bagade, Vijaya K. Shandilya, Use of histogram Equalization in Image Processing For Image Enhancement, International Journal of Software Engineering Research & Practices Vol.1, Issue 2, April, 2011

[7] Fabrizio Russo, an Image Enhancement Technique Combining Sharpening and Noise Reduction, IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, VOL. 51, NO. 4, AUGUST 2002

[8]RadhikaSivaramakrishna, Nancy A. Obuchowski, William A. Chilcote, Gilda Cardenosa, Kimerly A. Powell, Comparing the Performance of Mammographic Enhancement Algorithms: A Preference Study, AJR: 175, July 2000

[9] Sudhamony S, Binu P J, Satheesh G, IssacNiwas S,SudalaimaniC,Nandakumar K, Muralidharan V and Baljit SinghBedi, Nationwide Tele-Oncology network in India - A framework for implementation, 2008 10th IEEE Intl. Conf. on e-HealthNetworking, Applications and Service (HEALTHCOM 2008) [11]http://docs.opencv.org/trunk/doc/py_tutorials/py_imgproc/py_histograms/py_histogram_equalization/py_histogram_equalization.html