A Project Submitted for The Degree of B. Sc. Of
Cyber Security & Data Analytics
Engineering

KernelLover: Hypervisor-Based AV Solution

Supervised by:

Prof. Nirmeen El-Bahnasawy
Assis. Prof. Abdalla Moustafa

By

Omar Ahmed M. Shehata

Aml Alaa El-dien Hussien **Rawan Mohammed Hanafy**

2023 - 2024

**A Project Submitted for The Degree of B. Sc. Of
Cyber Security & Data Analytics
Engineering**

Faculty of Electronic Engineering
Menoufia University, Egypt

Approved by

1. **Prof.**
2. **Prof.**
3. **Prof.**
4. **Prof.**

9th July 2024

Acknowledgement

We would like to express our deep gratitude to Prof. Abdalla Moustafa and Prof. Nirmeen el-Bahnasawy, who brought the weight to their considerable experience and knowledge to this project. Their high standards have made us better at what we do.

We would like to thank our families to support us through our journey and believe in us. Their love and support helped us a lot to achieve many goals.

We are grateful to everyone who offered us moral support, encouragement, and uplifted our spirits. Your contributions have made a significant difference in our journey.

Abstract

Traditional methods employed by antivirus (AV) vendors and Endpoint Detection and Response (EDR) solutions, such as user-mode API hooking, have shown limitations in effectively combating modern cyber threats. User-mode API hooking involves monitoring applications at the user level, which can be bypassed by sophisticated malware that operates at deeper system levels. This approach often falls short in providing comprehensive protection against advanced threats.

This thesis explores a transformative approach by advocating for the adoption of hypervisor-based AV solutions to enable kernel-level system call hooking. Unlike user-mode API hooking, kernel-level system call hooking involves monitoring and intercepting system calls at the core of the operating system. By operating at this deeper level, hypervisor-based AV solutions can detect and block malicious activities that traditional methods might miss.

The comprehensive analysis presented in this thesis delves into the technical intricacies of implementing kernel-level system call hooking through hypervisor-based architectures. It examines how hypervisors, which manage virtual machines and operate below the operating system, provide a robust platform for monitoring and controlling system activities. This low-level access allows hypervisor-based solutions to intercept system calls before they reach the operating system, offering a higher level of security.

Table of Contents

1

Introduction

The world of cyber security is a constant arms race, our digital defences must evolve to counter ever-more-sophisticated cyber threats. Traditional methods employed by antivirus (AV) software and Endpoint Detection and Response (EDR) systems are like the old gatekeepers who meticulously checked everyone's identification at the castle entrance (user-mode API hooking). While this approach stopped many unwanted visitors, it had limitations. Clever attackers could forge documents or exploit weaknesses in the checking process (advanced malware). This thesis proposes a revolutionary approach to fortify our digital castles: hypervisor-based AV with kernel-level system call hooking.

Imagine a new kind of defence – not just guards at the gate, but ones with a complete view of the entire castle (kernel level). Hypervisor-based AV with kernel-level system call hooking acts like these omniscient guards. By operating at the core of the operating system (the kernel), it can monitor all the fundamental interactions happening within the system (system calls). This grants a much deeper understanding of what's going on, similar to the guards seeing everything happening inside the castle walls. This comprehensive view allows us to detect suspicious activities much more effectively, just as the guards with a full view could easily spot intruders hiding in the shadows. Traditional AV methods, like the gatekeepers, might miss these hidden threats because their view is limited. This new approach, with its ability to see everything happening at the most critical level, significantly enhances our ability to detect and combat modern cyber threats.

2

Intel protection Rings

Protection Rings are hierarchically arranged protection domains that are used in computer science to improve fault tolerance and provide computer security. Operating systems offer varying levels of access to resources, with rings arranged from most privileged to least privileged. The use of protection rings creates logical space for permission levels and execution levels. Two major applications of protection rings are: Increasing Fault Tolerance; Providing Computer Security; There are four levels in total, with level 0 being the most privileged and level 3 being the least privileged. Most Operating Systems use level 0 for the kernel or executive and level 3 for application programs. A resource that is accessible to level n is also accessible to levels 0 to n and the privilege levels are rings.

2.1 Protection Ring Modes

Supervisor Mode and Hypervisor Mode are the two main modes. These are briefly discussed as follows below.

1) **Supervisor Mode**: In certain processors, Supervisor Mode is an execution mode that permits the execution of all instructions, including privileged ones. Additionally, it provides access to various peripherals, memory management devices, and address spaces. The operating system typically operates in this mode.

Most processors have at least two different modes. The x86-processors have four different modes divided into four different rings. Programs that run in Ring 0 can do *anything* with the system, and code that runs in Ring 3 should be able to fail at any time without impact to the rest of the computer system. Ring 1 and Ring 2 are rarely used, but could be configured with different levels of access.

2) **Hypervisor Mode**: To regulate "Ring 0" hardware access, hypervisors use x86 virtualization instructions that are available on modern CPUs. To aid in virtualization, Intel VT-x(codenamed “Vander pool)” and AMD-V(codenamed “Pacifica”) have added nine new "machine code" instructions that are limited to Ring −1 and are meant to be utilized by hypervisors, as well as a new privilege level below “Ring 0”.

Both VT-x and AMD-V allow a guest operating system to run Ring 0 natively without affecting other guests or the host OS.

Before hardware-assisted virtualization , guest operating systems run under ring 1. Any attempt that requires a higher privilege level to perform (ring 0) will produces an interrupt and then handled using software, so called "Trap and Emulate".

2.2 Implementation

In certain systems, protection rings are used in conjunction with processor modes to combat the slave rule. Operating systems on hardware compliant with these regulations have the option to employ both security measures or only one of them. Close communication between the operating system and hardware is necessary for the effective utilization of Protection Ring architecture. Because of the way the operating system is built, it has been tested on numerous platforms and might use a different implementation mechanism on each one. Even if hardware offers more granularity in performance levels, the security model is typically reduced to two levels of access: the "core" level and the "user" level. The Protection Ring's features include:

1) Protection Ring follows hierarchy.
2) Protection Ring provides layered architecture.
3) Protection Ring provides Computer Security.

Good Fault Tolerance is provided by the Protection Ring.

2.3 Benefits of Safety Chains

Utilizing Protection Rings offers improved control over system resource access, which is one of its main benefits. It enhances computer security by guaranteeing that users can only access resources to which they are allowed. Furthermore, by lowering the possibility of errors, the hierarchical design of rings increases the fault tolerance of the system.

2.4 Protection Ring Drawbacks

Adding protection rings can increase the complexity of the system, which makes it more difficult to maintain and operate. This is one of the key drawbacks of protection rings. It may also result in poorer performance because of the extra layers of security and processing involved.

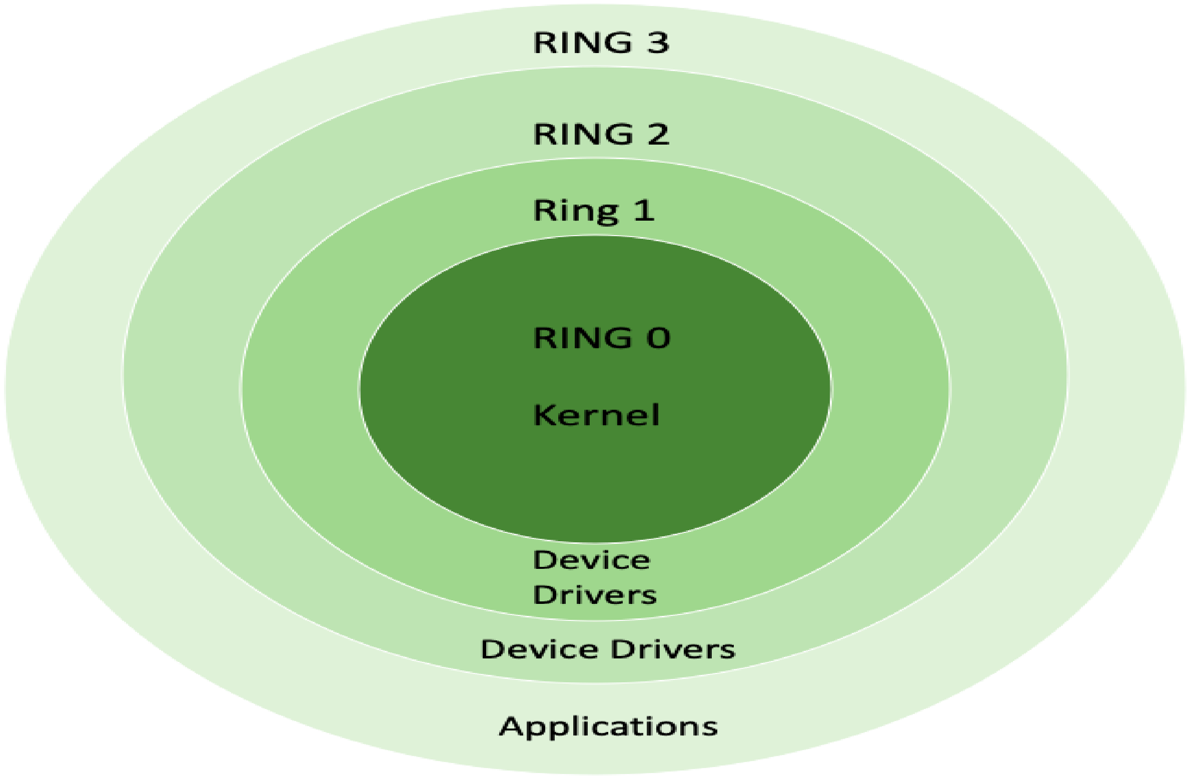


Fig 2.1 Privilege rings for the x86 available in protected mode

A privilege level in the x86 instruction set controls the access of the program currently running on the processor to resources such as memory regions, I/O ports, and special instructions. There are 4 privilege levels ranging from 0 which is the most privileged, to 3 which is least privileged. Most modern operating systems use level 0 for the kernel/executive, and use level 3 for application programs. Any resource available to level n is also available to levels 0 to n, so the privilege levels are rings. When a lesser privileged process tries to access a higher privileged process, a general protection fault exception is reported to the OS.

It is not necessary to use all four privilege levels. Current operating systems with wide market share including Microsoft Windows, macOS, Linux, iOS and Android mostly use a paging mechanism with only one bit to specify the privilege level as either Supervisor or User (U/S Bit).Windows NT uses the two-level system .The real mode programs in 8086 are executed at level 0 (highest privilege level) whereas virtual mode in 8086 executes all programs at level 3.

Potential future uses for the multiple privilege levels supported by the x86 ISA family include containerization and virtual machines. A host operating system kernel could use instructions with full privilege access (kernel mode), whereas applications running on the guest OS in a virtual machine or container could use the lowest level of privileges in user mode. The virtual machine and guest OS kernel could themselves use an intermediate level of instruction privilege to invoke and virtualize kernel-mode operations such as system calls from the point of view of the guest operating system.

3

Classic Anti-Virus Design

A program called an antivirus product is made to find and eliminate viruses and other types of harmful software from your laptop or computer. Malware, sometimes referred to as malicious software, is computer code that can damage your laptops and PCs as well as the data they hold. Malware hidden on a USB drive, in an attachment linked to a questionable email, or even just by visiting a questionable website can all unintentionally infect your devices.

Antivirus software's main job is to find, detect, and coordinate the removal of malicious software from a computer. Traditionally, this procedure entails comparing a computer's files and programs to a database of known malware types in order to spot any suspicious behaviour that might point to infection. When malicious code is found, the antivirus program neutralizes the threat by either deleting the offending files from the system or placing them in quarantine.

Conventional Antivirus: The Recognized Method Conventional antivirus software, which has long served as the cornerstone of digital defence, mainly uses signature-based detection. Using a database of recognized signatures, this technique uses malware identification. This method works well against known viruses that are widely distributed, but it has trouble identifying newly created, altered, or undiscovered malware.

Advantages of Conventional Antivirus:

1. Proven efficacy against known threats: Conventional antivirus software excels in identifying and eliminating malware that has been previously detected and categorized within its signature database.
2. Straightforward operation: Traditional antivirus relies on signatures, which makes it a user-friendly and manageable solution for both individuals and businesses. This approach to malware detection is straightforward.
3. Cost-effectiveness: A lot of conventional antivirus programs are "add-ons" or integrated into other goods. Thus, without requiring a payment, traditional antivirus software can offer a minimal level of protection.

3.1 Solution Design

For decades, antivirus software has been the cornerstone of digital defense. It's the familiar shield that stands guard against malware and viruses, protecting our computers from a barrage of digital nasties. However, the cyber security landscape is constantly evolving, and traditional antivirus is struggling to keep pace with increasingly sophisticated threats.

explores the limitations of traditional antivirus and introduces a more robust solution: Endpoint Detection and Response (EDR) systems. We'll delve into the design philosophies behind both approaches and highlight the reasons why EDR is becoming the new standard for endpoint security.

3.1.1 The Antivirus Workhorse: A Signature-Based Approach

Traditional antivirus software relies on a technique called "signature-based detection." Imagine a bouncer at a club checking IDs against a list of known troublemakers. The antivirus software maintains a database of digital fingerprints, or signatures, of known malware. When it scans a file, it compares the file's signature against the database. If there's a match, the antivirus identifies the file as malicious and takes action to quarantine or remove it.

In IT security, EDR stands for Endpoint Detection and Remediation. This kind of security software keeps an eye on threats to computer networks and takes appropriate action. EDR tools have the ability to identify harmful activity, such as malware infections or odd user behavior, and then take appropriate action to eliminate the threat. Businesses of all sizes use EDR tools to safeguard their networks from attacks. Certain EDR tools, like those for on-premises or cloud-based networks, are made specifically for those kinds of networks.

Others are applicable to all kinds of networks. A monitoring system, a detection system, and a response system are the three primary parts of an EDR tool. Data is gathered by the monitoring system from the network's computers. After analyzing the data, the detection system searches for indications of malicious activity. The response system moves to neutralize or eliminate the threat. Various approaches can be taken to deploy EDR tools, contingent on the requirements of the company. They can be set up on personal computers, servers, or cloud storage. While some EDR tools are sold separately, others are a part of a more comprehensive security package. EDR solutions are used by organizations to defend their networks against a range of dangers, such as malware, phishing scams, and insider threats. EDR tools are also useful for adhering to legal requirements, like the General Data Protection Regulation (GDPR).

The primary benefits of using EDR software include the ability to detect and respond to threats in real-time, reduce the time between detection and response, and minimize the impact of a security-incident.

3.1.2 Traditional Antivirus vs. EDR

Some of the key differences between EDR and traditional antivirus are   discussed below:

Traditional Antivirus

In comparison to modern EDR systems, traditional antivirus programs are
more restricted and simplistic. It is possible to think of antivirus software as a component of the EDR. The majority of the time, antivirus software is a single program with the ability to

scan for, identify, and eliminate viruses and other malware. Enterprise virus protection for any endpoints that the antivirus is installed on can be obtained with an enterprise-wide antivirus program. Consider that EDR is more important for safeguarding the endpoints in your company when weighing internet security vs. antivirus software. Nonetheless, the EDR security system has a far wider purpose. Several security tools, such as firewalls, whitelisting tools, monitoring tools, etc., are also included in EDR in addition to antivirus software. To offer all-encompassing defence against online hazards. It often utilizes the client-server architecture and maintains the security of the various endpoints on a business's digital network. Given that traditional antivirus software is no longer a reliable source of complete security, EDR security solutions are a better fit for today's businesses.

Cybercriminals are becoming more adept and smarter at their trade and using advanced threats to breach networks. Traditional antiviruses provide you with a basic level of protection from such advanced cyber-attacks and are not sufficient to meet your network security needs. A traditional antivirus program detects malware and viruses by signature-based detection which is loaded into its database. However, hackers are now capable of creating malware with continuously evolving codes that can easily bypass traditional antiviruses. EDR systems detect all endpoint threats and provide real-time responses to the identified threats. It can help you understand the complete scope of the potential attack which increases your preparedness for such attacks. EDR systems also collect high-quality forensic data which is needed for incident response and investigations. Overall, EDR security systems are much better equipped to handle cyber threats than traditional antivirus.

Anti-Virus Software Structure:

Anti-virus software has various forms depending on the functions and operating methods. (Fig. 3.1) shows an anti-virus structure that provides real-time and user-executed malware detection, restoration, and access denial to malware functions when the original files are un-restorable. In general, anti-virus software consists of four components. The Anti-virus Manager handles overall management functions. It is a user interface that can execute malware scans, review the results of malware restoration, raise alarms, and set a periodic scan policy for malware. In an enterprise environment, the Anti-virus Manager is an indirect interface for integrated management. The security policy set by the security administrator is enforced and the anti-virus operation information requested by the security administrator is transmitted. The Anti-virus Manager also provides a function to update the Malware Signature DB and the Anti-virus Engine, which is responsible for malware inspection and treatment . The Anti-virus Engine is a collection of technical mechanisms for handling malware in the system. It extracts file information and compares it with the Malware Signature DB to determine whether the file is malware. If a file is malware , the Anti-virus Engine restores it. If the file cannot by restored, the Anti-virus Engine creates an access denial policy and sends it to the Anti-virus Driver to enforce. In the Malware Signature DB, malware signatures for determining whether a scanned file contains malware and rules for restoring malware-infected files are registered .

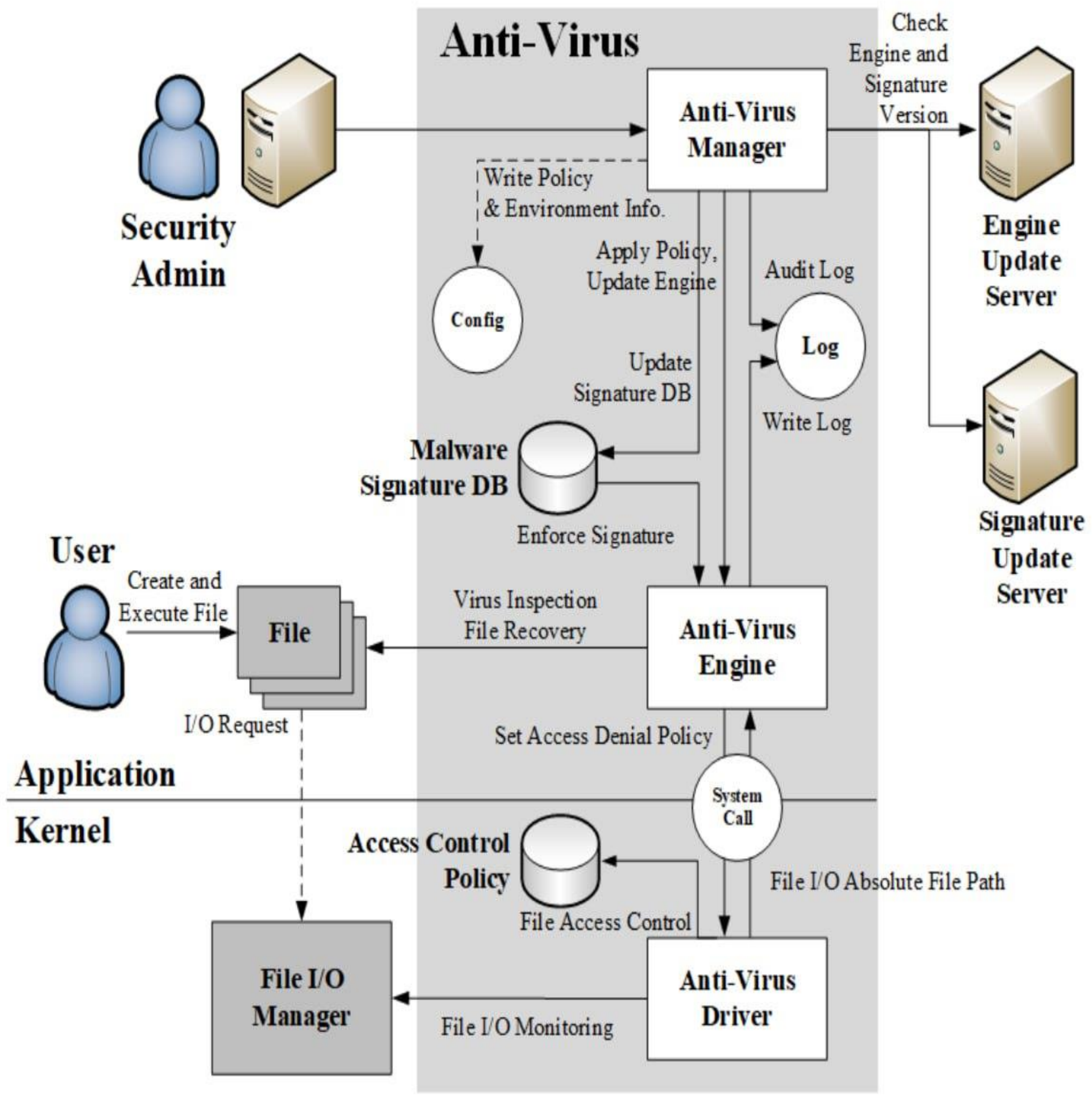


Fig 3.1. General structure of Anti-virus software

The Anti-virus Driver is a real-time malware entry monitoring function that denies access to malware-infected user files. It monitors the I/O of the file system and retrieves the absolute file path for the Anti-virus Engine to check for malware . When the Anti-virus Engine determines that a file is malware but cannot restore the file, the Anti-virus Driver receives an access denial policy created and delivered by the Anti-virus Engine; it registers and enforces the policy to deny user access.

3.2 Problems with the design

This approach works well for known threats – the tried and tested malware variants with well-documented signatures. However, it has significant limitations:

- Blind Spots: New and evolving malware can easily bypass signature-based detection. Hackers can modify existing malware or create entirely new variants with no known signatures, rendering traditional antivirus vulnerable.
- Limited Visibility: Antivirus software primarily focuses on user-mode interactions, where programs interact with the operating system. This leaves a blind spot for threats that operate at a deeper level, like those exploiting vulnerabilities in the system kernel.
- Slow to Adapt: Since antivirus relies on having signatures for known threats, it can be slow to react to entirely new attacks. It's like having to print and distribute new wanted posters every time a new criminal appears.

3.2.1 Traditional Anti-Virus's Drawbacks

1) **Incapacity to Handle Zero-Day Attacks:**

Before they can be fixed or a signature is made, zero-day attacks take advantage of undiscovered vulnerabilities, leaving systems vulnerable to fresh threats. Traditional antivirus software finds it difficult to defend against these attacks.

2) **Absence of Behavioral Analysis:**

Conventional antivirus solutions, in contrast to next-generation antivirus software, do not have the ability to perform behavioral analysis, which prevents them from identifying malware based alone on unusual or suspicious conduct on the system.

3) Dependency on Frequent Updates:

The malware signature database of traditional antivirus software is updated frequently, which is essential to its efficacy.

Security is put at risk when software is not updated on a regular basis because it becomes less effective at identifying and stopping assaults.
Restricted to Malware Detection: Conventional antivirus software frequently just concentrates on the identification and elimination of malware, without providing extra security measures , such as phishing protection, network attack defence, or encryption, which are increasingly necessary in today’s complex cyber threat landscape.

4

Intel VTX

4.1 Intro to intel Virtualization

The Intel Virtualization Technology is known as Vt-x. and formerly known as Vanderpool Additionally , this makes it easier for a computer to run numerous operating systems at once. It is known as Vt-x in Intel processors, however AMD processors will use AMD-V or Hyper-V (especially in Ryzen CPUs) technology instead. All of these are therefore products of virtualization technology, which enables us to run multiple OS instances simultaneously on our system. For instance, we must enable it in order to use VMware or VirtualBox. Because we can practically install and run different operating systems on these kinds of software. Here, we just install the operating systems virtually in VMware or VirtualBox and run them simultaneously and independently rather than installing them on our hard drive.

Through virtualization, physical infrastructure and computing environments are divided, allowing a single physical computer or server to be divided into multiple virtual computers. Multiple workloads can operate in complete isolation from one another on shared virtualized hardware with good performance.

Client virtualization is used by many companies today for these and other benefits:

the capacity to run various operating systems or altered iterations of the same OS. For instance, this allows a coder to write code on the same computer under both Linux and Windows 10. separating tasks on the same computer to improve system resilience, security, and uptime. Isolated workspaces can be launched on any suitable PC thanks to virtualization. A workspace's influence on the others is reduced because they are separate areas.

Benefits of Virtualization:

- **Improved system uptime:** Virtual machines can be easily restarted or restored if something goes wrong.
- **Enhanced protection for digital assets:** Virtualization allows isolating critical systems from potential threats.
- **Stronger business continuity:** If a physical machine fails, virtual machines can be quickly migrated to another machine to keep operations running.

4.1.1 Virtualization Protection Using Intel® Hardware-Based Security:

Virtualized workloads benefit greatly from the additional levels of security provided by hardware-based security features. Because it operates at the system level, embedded hardware security technology has a more comprehensive understanding of the computer environment than software-based security solutions. Hardware-based security features provide a multifaceted strategy that incorporates security layers below the operating system to complement standard software-based tools.

The frequency of cyber-attacks is rising. Restoring system functionality as soon as feasible is crucial when attacks happen. Virtualization contributes to resilience by assisting in the defense against malware of the primary OS kernel, applications, and secrets. In order to improve security, data privacy, and compliance, virtualization also makes it possible to create numerous separated work environments. For example, this feature might be used to separate work from personal activities. Hardware-enforced virtualization security primitives are combined with software-based security solutions by Intel hardware security to build virtual machines that are reliable and efficient.

4.1.2 Intel® Virtualization Technology for Connectivity:

Each new generation of processor brings more processing power and new capabilities to server platforms, and today's servers are running more applications and processes simultaneously than ever before. IT departments are deploying 10 Gigabit Ethernet (10GbE) to meet the I/O demands of these servers and also to support converged LAN and SAN networking. 10GbE's increased throughput and support for multiple traffic types gives IT administrators the opportunity to simplify their network infrastructures and increase network flexibility to adapt to changes in demand quickly and effectively. Intel® Virtualization Technology for Connectivity1 (Intel® VT-c) is a key feature of many Intel® Ethernet Controllers. With I/O virtualization and Quality of Service (QoS) features designed directly into the controller's silicon, Intel VT-c enables I/O virtualization that transitions the traditional physical network models used in data centers to more efficient virtualized models by providing port partitioning, multiple Rx/Tx queues, and on-controller QoS functionality that can be used in both virtual and non-virtual server deployments.

4.1.3 Quality of Service for On-Controller Systems in Virtualized and Non-Virtualized Environments :

The core of Intel VT-c consists of technologies and functions that are integrated into the Intel Ethernet Controller to provide a common QoS feature set. This feature set delivers a range of capabilities that can be customized by an IT administrator to meet specific needs, or they can be used directly by an operating system or hypervisor (Figure 1).
Two of the optimization technologies used to support the expanded I/O virtualization functionalities of Intel VT-c are Virtual Machine Device Queues (VMDq) and PCI-SIG* Single Root I/O Virtualization (SR-IOV).
By shifting functionality to the Intel Ethernet Controller, these features aid in the reduction of I/O bottlenecks and enhance overall server performance. Numerous QoS characteristics built into the silicon of the controller are shared by them, and Native throughput, equitable bandwidth distribution, and enhanced I/O scalability are all offered by them.
These features relieve I/O bottlenecks and enhance overall server performance in virtualized servers by shifting the Intel Ethernet Controller's data sorting and queuing functions from the hypervisor.

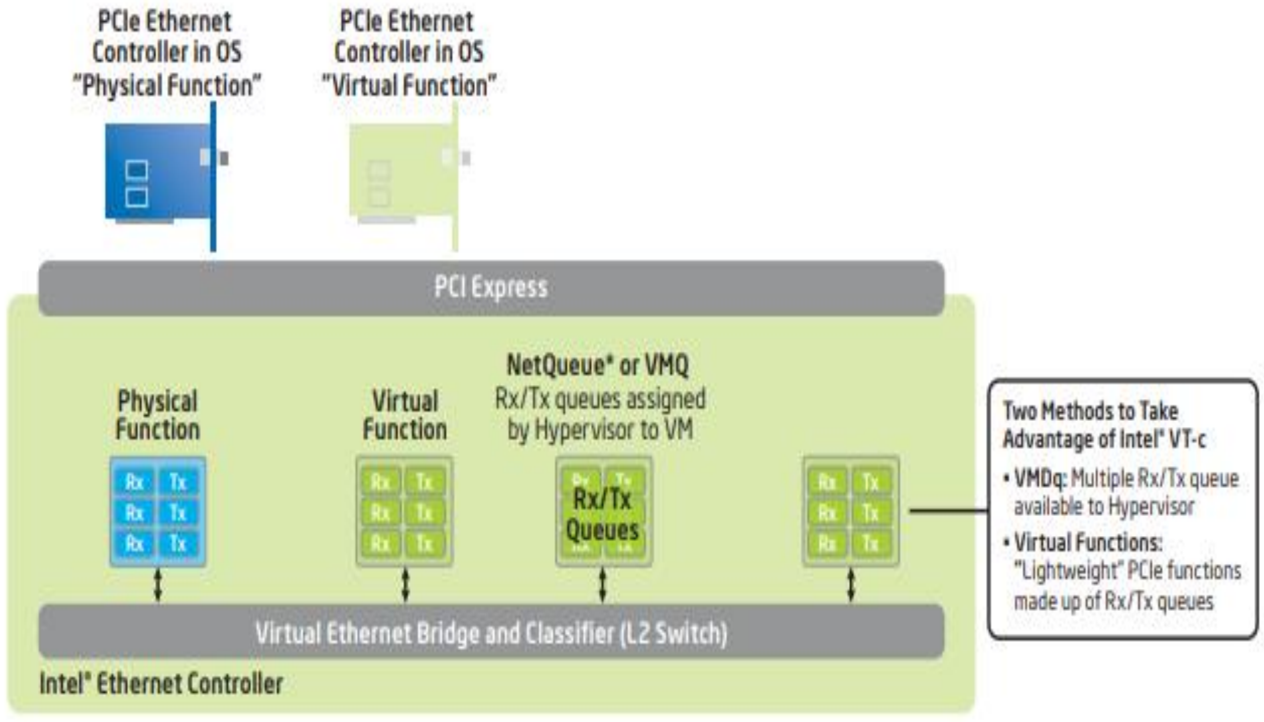

PCIe Ethernet Controller in OS "Physical Function"
PCIe Ethernet Controller in OS "Virtual Function"
PCI Express
Physical Function
Virtual Function
NetQueue* or VMQ
Rx/Tx queues assigned by Hypervisor to VM
Rx Tx
Rx/Tx Queues
Two Methods to Take Advantage of Intel® VT-c
• VMDq: Multiple Rx/Tx queue available to Hypervisor
• Virtual Functions: "Lightweight" PCIe functions made up of Rx/Tx queues
Virtual Ethernet Bridge and Classifier (L2 Switch)
Intel® Ethernet Controller

Fig4.1: overview of intel virtualization technology for connectivity

In non-virtualized environments, port partitioning provides hardware based QoS features that enable a physical port to consolidate the traffic of a greater number of physical ports with no loss of functionality and more flexible bandwidth allocation. Both technologies enable a single Ethernet port to appear as multiple adapters to virtual machines (VMs) by allowing the Intel Ethernet Controller to place data packets directly into individual VM memory stacks using a process called direct memory access (DMA). Each VM’s device buffer is assigned a transmit/receive queue pair in the Intel Ethernet Controller, and this pairing between VM and network hard ware helps avoid needless packet copies and route lookups in the virtual switch. The result is less data in the host server’s buffers and an overall improvement in I/O performance.

4.1.4 Hypervisor-controlled Port Partitioning Using Virtual Machine Device Queues:

VMDq works in conjunction with VMware Net Queue* or Microsoft Virtual Machine Queues* (VMQ), in their respective hypervisors, to use the sorting and queuing functionality in the controller for traffic steering and Tx/Rx round-robin scheduling for balanced bandwidth allocation across multiple transmit and receive queues. Using these technologies, With VMDq, the host's buffers hold less data and I/O operations perform better overall because the hypervisor can represent a single network port as several ports that are assigned to the virtual machines. When VMDq is turned on, the hypervisor manages queue assignment, providing on-controller QoS features and port partitioning advantages with little to no administrative work for the IT department (Figure 4.2).

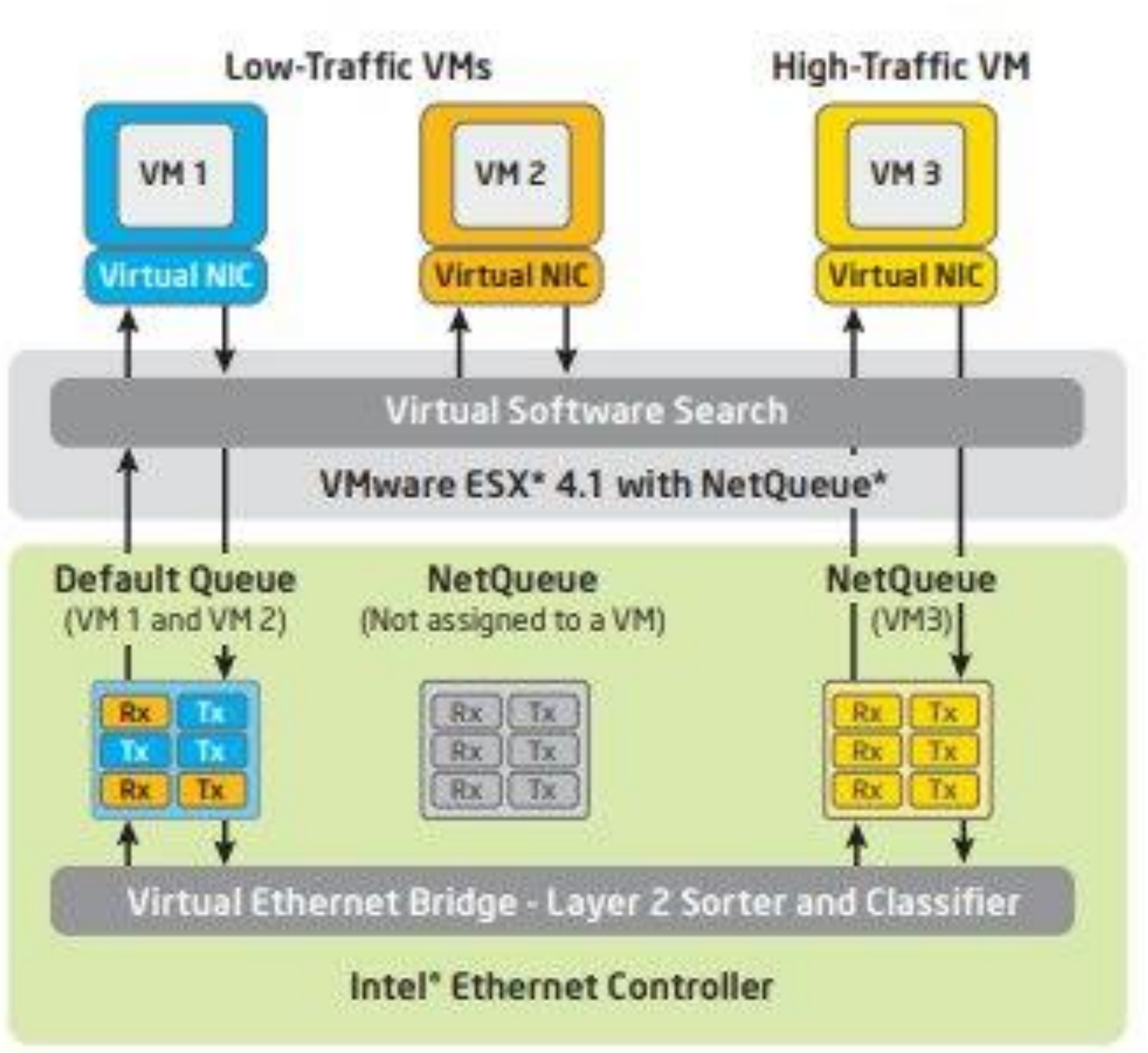


Fig4.2 Virtual Machine Device Queues

4.2 Advantages of Virtualization

The Software is used in virtualization to build an abstraction layer on top of the actual hardware. By doing this, it builds a system of virtual computers, or virtual machines (VMs). This effectively divides a single physical server into numerous virtual servers and enables enterprises to run multiple virtual machines, operating systems, and applications on it.

To put it simply, one of the primary benefits of virtualization is that it makes better use of the actual computer hardware, which increases return on investment for the business.

- **Cost Savings:** Virtualization can be like having several computers for the price of one. Instead of buying and maintaining multiple physical machines, you can run several virtual machines on a single computer, saving on hardware costs and electricity.
- **Flexibility:** Need to run a specific program that requires a different operating system? No problem! With virtualization, you can create a virtual machine with the exact operating system you need, all on the same physical machine. This gives you more flexibility to run different programs without needing dedicated computers.
- **Easier Management:** Think of it like managing multiple shops in one warehouse. Virtualization tools allow you to easily manage and control all your virtual machines from a single place. You can quickly start, stop, or restart virtual machines as needed, making it easier to maintain your computer systems.
- **Improved Uptime:** If something goes wrong with a virtual machine, it won't necessarily crash the entire system. You can simply restart the virtual machine without affecting the others. This helps to keep your important applications and services running smoothly.
- **Disaster Recovery:** What happens if a physical server gets damaged? It can take hours or even days to fix or replace it, bringing your business to a standstill. With virtualization, if a virtual machine gets affected, you can simply copy it to another machine and get things running again in minutes. Virtualization makes your systems more resilient to problems.
- **Boost Efficiency and Productivity:** With fewer physical servers to manage, your IT team can spend less time on maintenance and more time on important projects. Virtualization tools allow you to easily update and maintain all your virtual machines from a single place, saving them time and headaches. A happy and efficient IT team means a smoother running business for everyone.

- **Empower Your Developers:** Virtualization lets you create isolated virtual machines for development and testing. This means developers can quickly set up new environments to test software updates or new features without affecting the main system. It's like having a personal sandbox for each developer to experiment in, leading to faster development cycles and a more agile business.

Overall, virtualization is like having a more efficient and flexible way to use your computer resources. It can save you money, improve manageability, and help ensure your critical systems stay up and running.

4.3 Virtualization Types

Businesses of all sizes are using virtualization as a way to lower IT expenses, enhance security, and boost operational effectiveness. Which one would work best for your company? We'll go over some of the most popular virtualization strategies here, along with how they might change your business.

1. Desktop Virtualization

With desktop virtualization, operating systems can be remotely stored on a server in a data centre. From any system, at any location, the user can virtually access their desktop. The primary benefits of desktop virtualization are user mobility, portability, and easy management of program installation, updates, and patches.

2. Network virtualization

Network virtualization turns all of the physical networking hardware into a single software-based resource. Additionally, it divides the bandwidth that is available into numerous independent channels that can be dynamically assigned to servers and other devices. Network virtualization may be advantageous for enterprises with a large user base that must always keep their systems up and running. Distributed channels will cause your network performance to soar, enabling you to deliver services and applications faster than ever.

3. Application Virtualization

A user can utilize application virtualization to access a server-based application from a distance. The application may still be used locally via the internet on a workstation, but the server retains all of the user's personal information and other features. An example of this would be a user who has to run two different versions of the same program. Applications that make use of application virtualization include packaged and hosted applications.

4. Storage Virtualization

This kind of virtualization is reasonably easy and affordable to set up since it calls for integrating your physical hard drives into a single cluster. Storage virtualization is helpful for disaster recovery planning because it allows data stored on your virtual storage to be replicated and moved to a different location. Reducing the hassles and costs associated with many storage systems by combining your storage into one.

5. Hardware Virtualization

A virtual machine management (VM) referred to as a "hypervisor" enables hardware virtualization, which is conceivably the most popular type of virtualization available today. To make the most use of all hardware resources, the hypervisor creates virtual clones of computers and operating systems and combines them into a single, sizable physical server. Additionally, it enables users to run several operating systems simultaneously on a single computer.

4.4 Intro to EPT

Intel® Extended Page Tables (EPT) is a hardware technology built into certain Intel processors that provides an extra layer of security for virtualized environments. Imagine you have a virtual apartment building (virtual machines) on top of a physical building (physical server). EPT acts like a high-security lock on each virtual apartment's door.

Here's how EPT helps tighten security in virtualization:

- **Stronger Memory Protection:** Virtualization lets multiple virtual machines share the same physical server. EPT creates a separate memory space for each virtual machine, like giving each apartment its own set of keys. This prevents one virtual machine from accidentally accessing another's memory, which could be a security risk.
- **Improved Malware Defence:** Some malware tries to tamper with a computer's memory to gain access to sensitive information. EPT acts like a guard checking everyone who enters the virtual apartment. It can identify and block unauthorized access attempts, making it harder for malware to infect the system.

- **Smoother Performance:** While providing extra security, EPT is designed to be efficient. It works with the processor to translate memory addresses quickly, minimizing any impact on virtual machine performance.

4.4.1 Benefits of Using EPT:

- **Enhanced Security:** EPT adds an extra layer of protection for virtualized environments, making it harder for attackers to gain access to sensitive data.
- **Improved Reliability:** By isolating virtual machine memory, EPT helps prevent conflicts and unexpected behaviour, leading to a more stable and reliable virtualized environment.
- **Peace of Mind:** Knowing your virtual machines have an extra layer of security can give businesses peace of mind when using virtualization for critical applications.

4.4.2 Who Uses EPT?

EPT is particularly valuable for organizations that rely heavily on virtualization, such as:

- **Cloud Service Providers:** They can offer secure virtual machines to their clients with the added protection of EPT.
- **Data Centres:** They can consolidate multiple servers onto fewer physical machines while maintaining high security standards for virtualized applications.
- **Businesses with Sensitive Data:** EPT can provide an extra layer of security for virtual machines that store or process sensitive information.

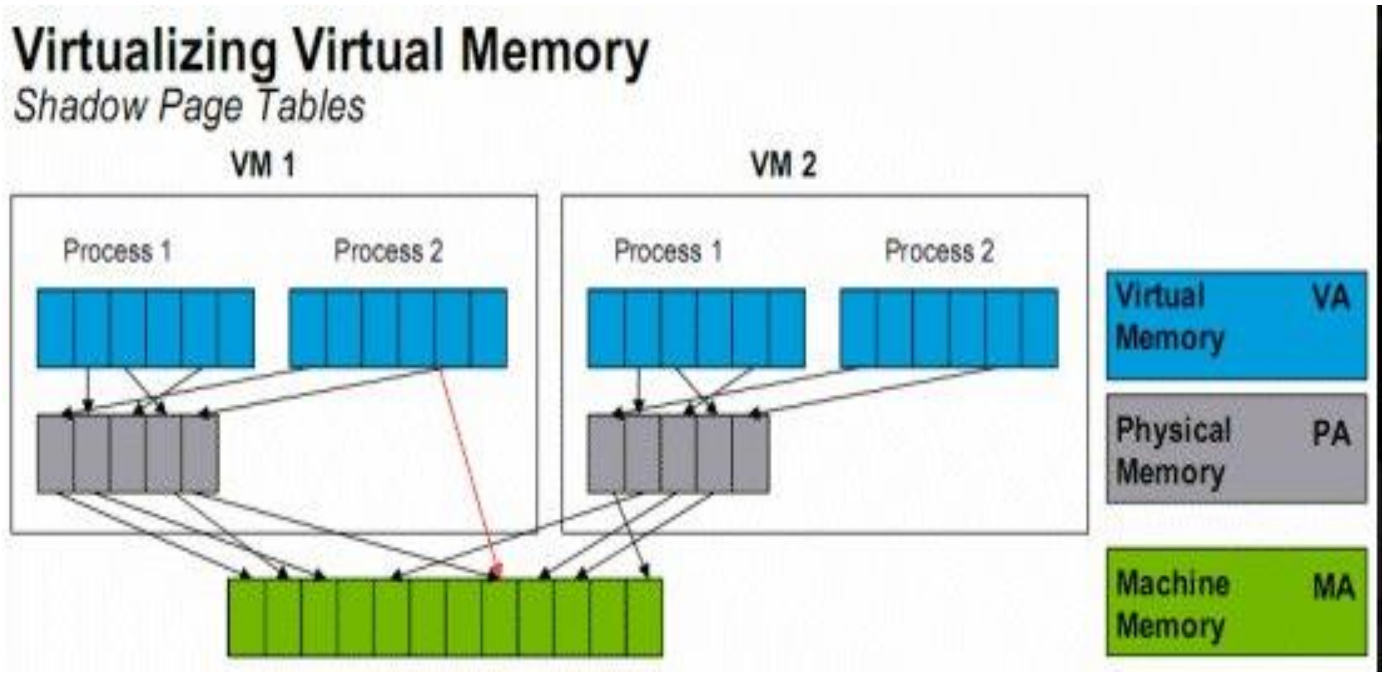


Fig 4.3 Virtualizing virtual memory

It would be necessary to convert virtual memory (blue) to "guest OS physical memory" (grey), and then back to real physical memory (green), if shadow pages weren't there. Fortunately, the "shadow page table" method avoids the double accounting by forcing the MMU to bypass the intermediate "guest OS physical memory" step and instead use a virtual memory (of the guest OS, blue) to real physical memory (green) page table. However, there is a catch: some "shadow page table" housekeeping is necessary for every update of the guest OS page tables. While this severely impairs the performance of the early hardware virtualization technologies, it is also detrimental to the performance of software-based virtualization solutions (BT and Para). The cause is that many of those really heavy VMexit and VMentry calls.

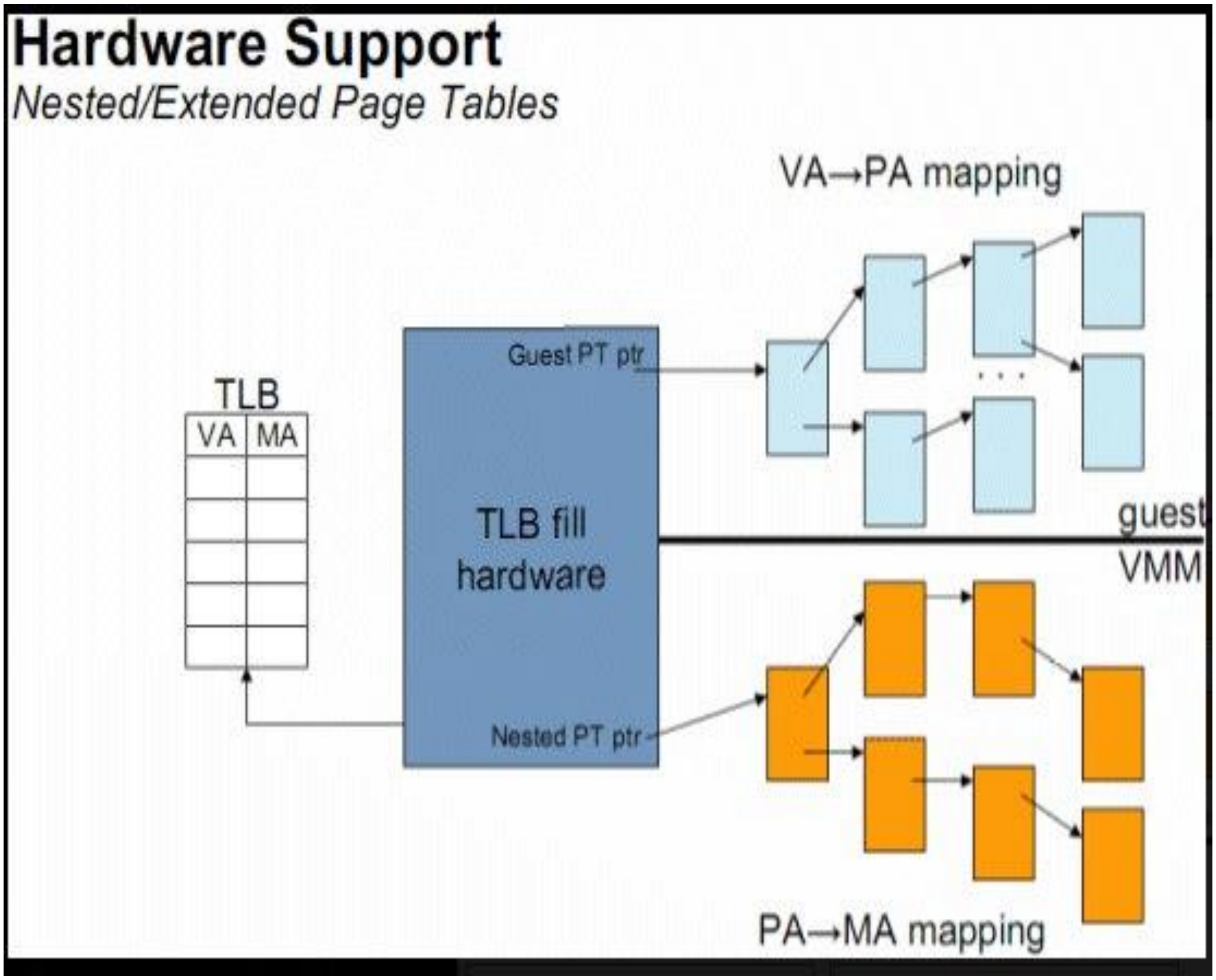


Fig4.4 EPT or Nested Page Tables is based on a "super" TLB that keeps track of both the Guest OS and the VMM memory management.

A CPU with hardware support for nested paging caches the transition from virtual memory (guest operating system) to physical memory (guest operating system) as well as the physical memory (guest operating system) to actual physical memory in the TLB, as seen in the above image. The Address Space Identifier (ASID), a new VM-specific tag, is present in the TLB. As a result, the TLB is able to maintain track of which TLB entry corresponds to which VM. A VM switch does not flush the TLB as a result. The TLB entries from the many virtual machines cohabit in harmony as long as the TLB is large enough.

This eliminates the need to continuously update the shadow page tables and greatly simplifies the VMM. If It is evident that layered paging can significantly increase speed (up to 23%, according to AMD) if we take into account that the Hypervisor must step in for each update of the shadow page tables (one for each active virtual machine). If each virtual machine (VM) contains more than one CPU, then nested paging becomes even more crucial. The shadow page tables must update much more frequently as a result of several CPUs having to frequently synchronize the page tables. The more (virtual) CPUs you employ per virtual machine (VM), the harsher the shadow page table performance penalty becomes. The CPUs merely synchronize TLBs as they would have in a non-virtualized environment while using stacked paging.

The sole drawback is that EPT, or nested paging, converts virtual to physical. If there is an incorrect entry in the TLB, address translation becomes much more complicated. We must complete every step in the orange region for every step we take in the blue area. As a result, 16 searches were conducted in the "native situation" instead of the original four table searches (four orange steps and four blue steps for each).

A CPU now need substantially larger TLBs than it did previously in order to compensate, and TLB misses are very expensive. In the event of a TLB miss in a native (non-virtualized) scenario, four searches in the main memory must be performed. Performance is then negatively impacted by a TLB miss. Examine the "virtualized OS with nested paging" TLB miss scenario now: sixteen (!) table searches are required.

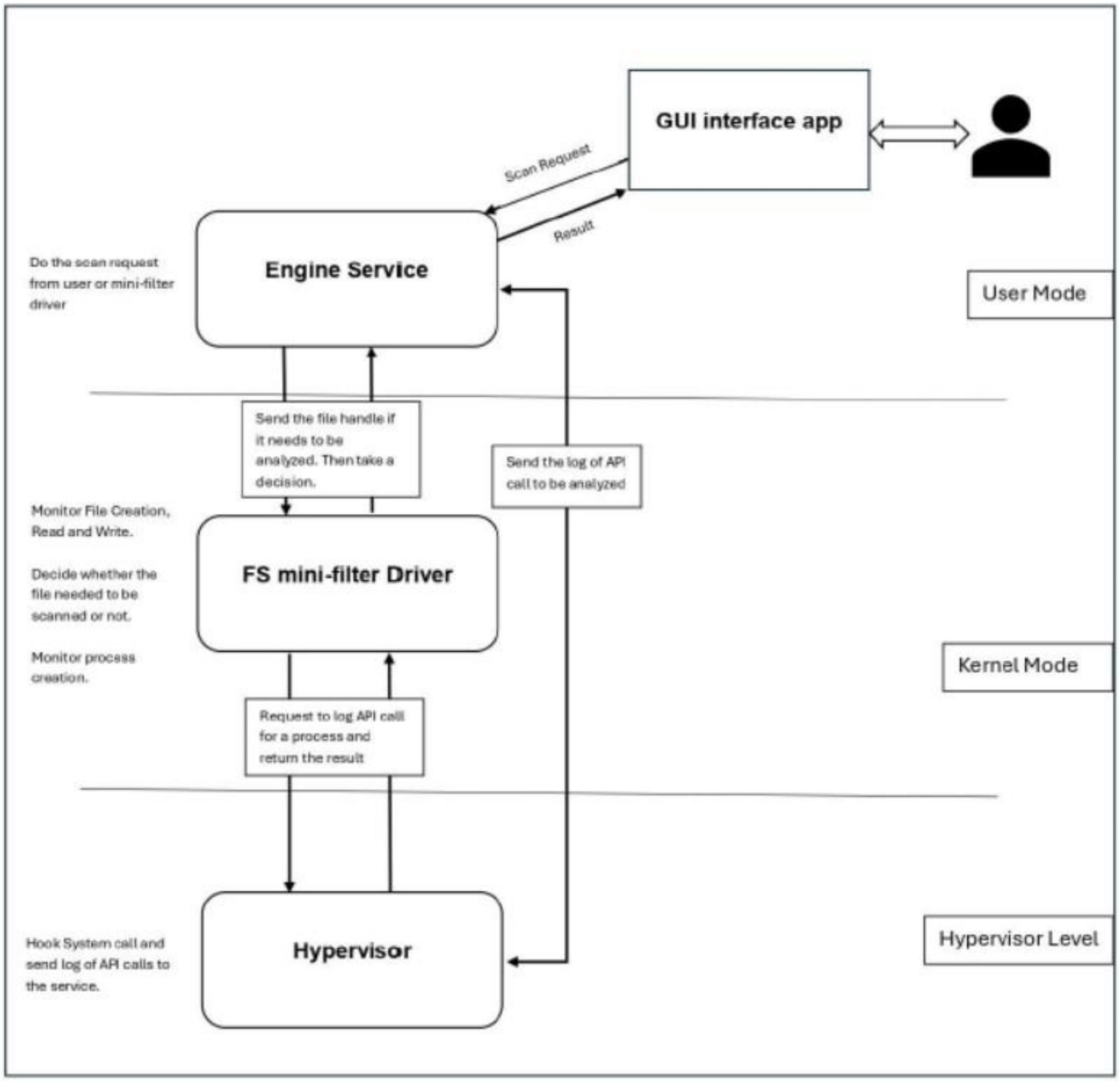
Situated in the system RAM with considerable latency. Our hit performance turns into a disastrous performance! Luckily, if the TLBs are rather large, only a small number of applications will result in a significant number of TLB misse

5

KernelLover Hypervisor-Based AV solution

5.1 Solution Design

The system is designed to provide robust security monitoring and scanning by integrating components that operate at different levels: User Mode, Kernel Mode, and Hypervisor Level. The primary components include a GUI Interface App, an Engine Service, a File-System Mini-Filter Driver, and a Hypervisor. This setup leverages the strengths of each level to ensure comprehensive and efficient threat detection and response.


GUI interface app
Scan Request
Result
Engine Service
Do the scan request from user or mini-filter driver
User Mode
Send the file handle if it needs to be analyzed. Then take a decision.
Send the log of API call to be analyzed
Monitor File Creation, Read and Write.
Decide whether the file needed to be scanned or not.
Monitor process creation.
FS mini-filter Driver
Kernel Mode
Request to log API call for a process and return the result
Hook System call and send log of API calls to the service.
Hypervisor
Hypervisor Level

5.1.1 User Mode: GUI Interface App and Engine Service

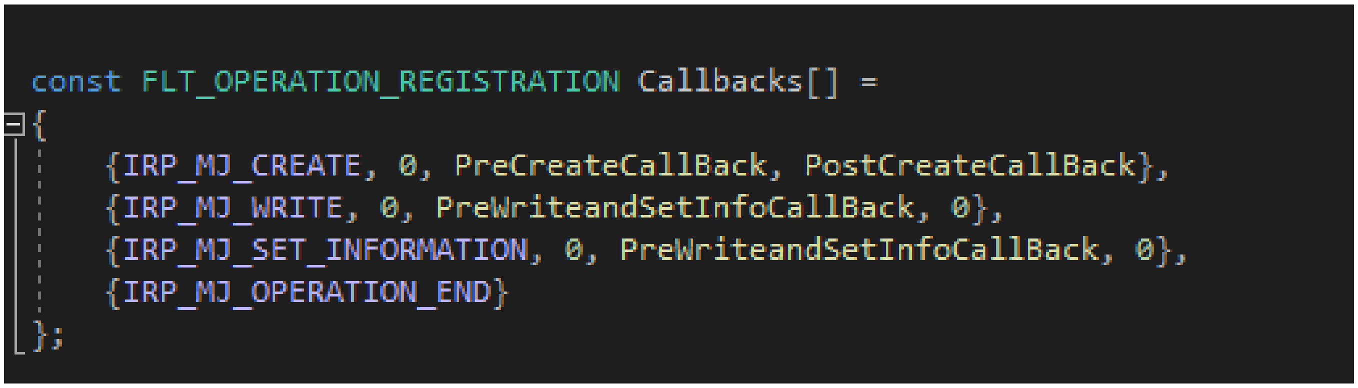
The user interacts with the system through the GUI Interface App. This application is the primary interface for users, allowing them to request scans of files, directories, or entire drives. It also provides the user with the results of these scans. When a user initiates a scan request, this request is sent to the Engine Service, a user-mode service responsible for scanning the target object and analyzing it for potential threats.

Upon receiving a scan request from the GUI Interface App, the Engine Service begins processing the request. It handles the initial examination of the target object and determines if further analysis is required. The Engine Service acts as a central coordinator, communicating with the Kernel Mode and Hypervisor Level components to ensure comprehensive scanning and monitoring.

5.1.2 Kernel Mode: FS Mini-Filter Driver

Operating at the kernel level, the File-System Mini-Filter Driver plays a critical role in monitoring file operations and process creation. This driver is always active, observing file creation, read, write operations, and new process creations. Its primary function is to decide whether a file needs to be scanned based on its observations.

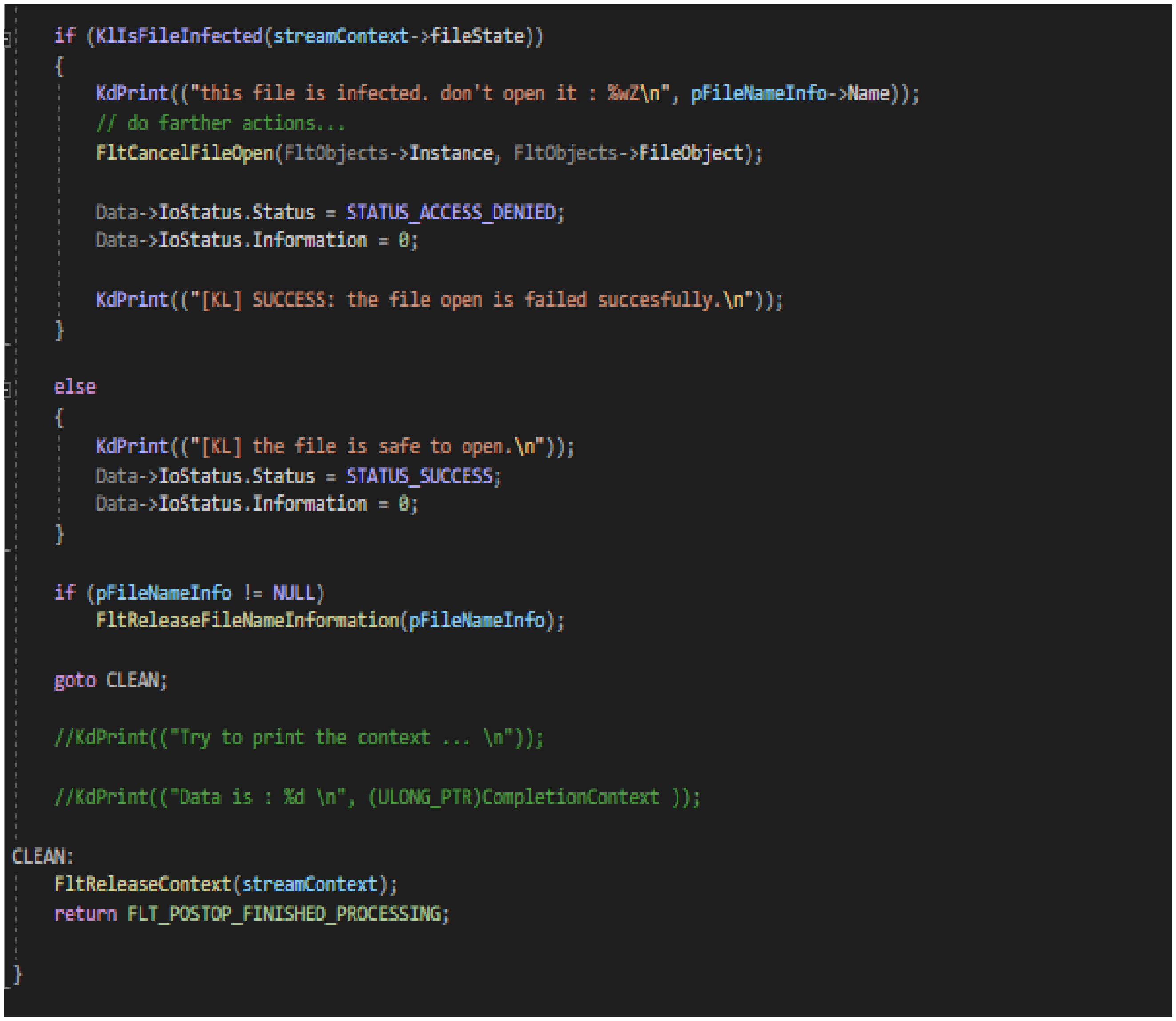
When the FS Mini-Filter Driver detects a file operation or process creation that requires attention, it communicates with the Engine Service. This communication includes sending logs of relevant API calls that need to be analyzed. The FS Mini-Filter Driver also requests the Hypervisor to log API calls for specific processes, enabling a deeper level of monitoring and



```
const FLT_OPERATION_REGISTRATION Callbacks[] =
{
    {IRP_MJ_CREATE, 0, PreCreateCallBack, PostCreateCallBack},
    {IRP_MJ_WRITE, 0, PreWriteandSetInfoCallBack, 0},
    {IRP_MJ_SET_INFORMATION, 0, PreWriteandSetInfoCallBack, 0},
    {IRP_MJ_OPERATION_END}
};
```

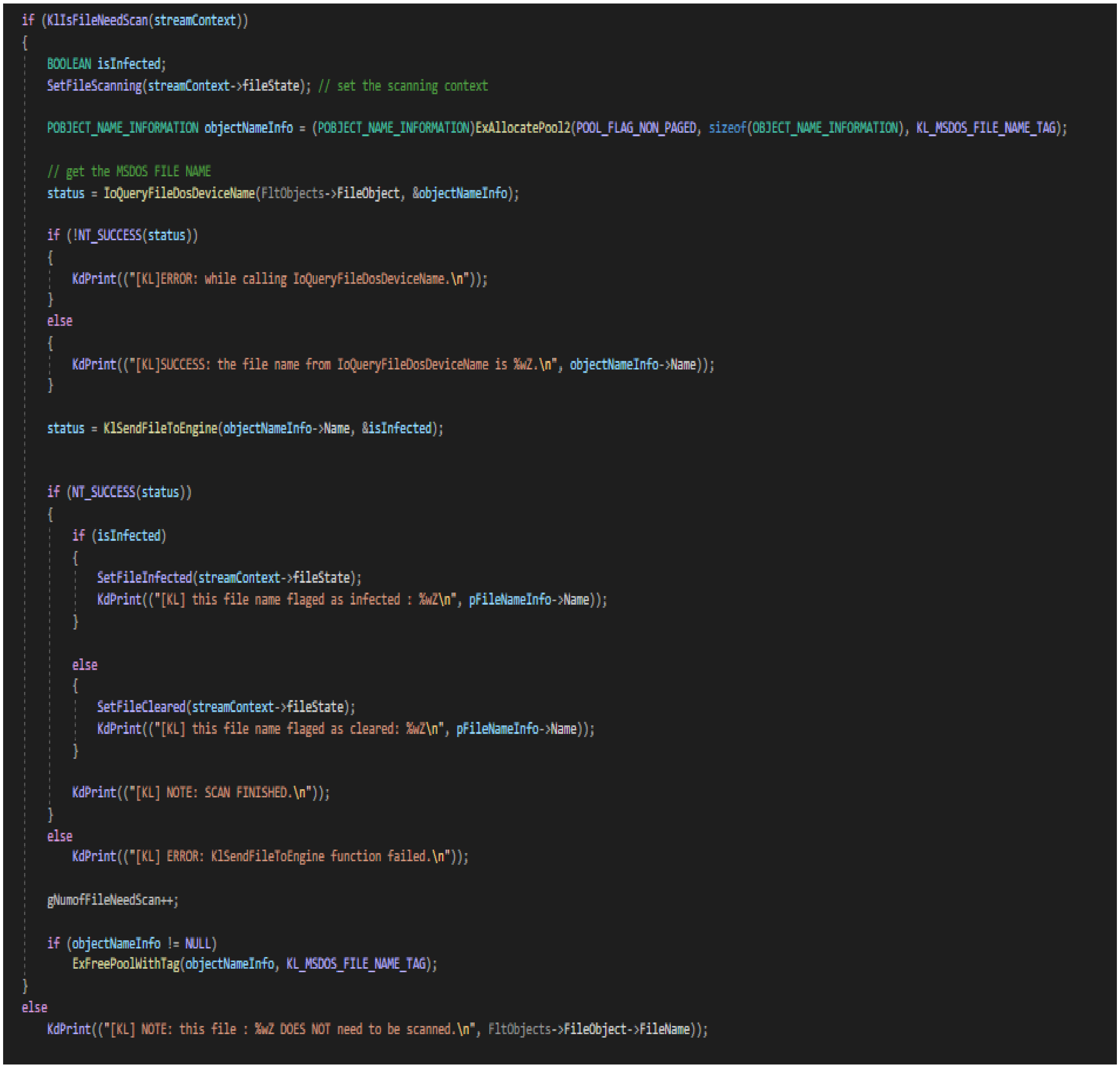
analysis.

First, I registered for pre and post routines for IRP_MJ_CREATE request, and a pre routine for IRP_MJ_SET IRP_MJ_SET_INFORMATION

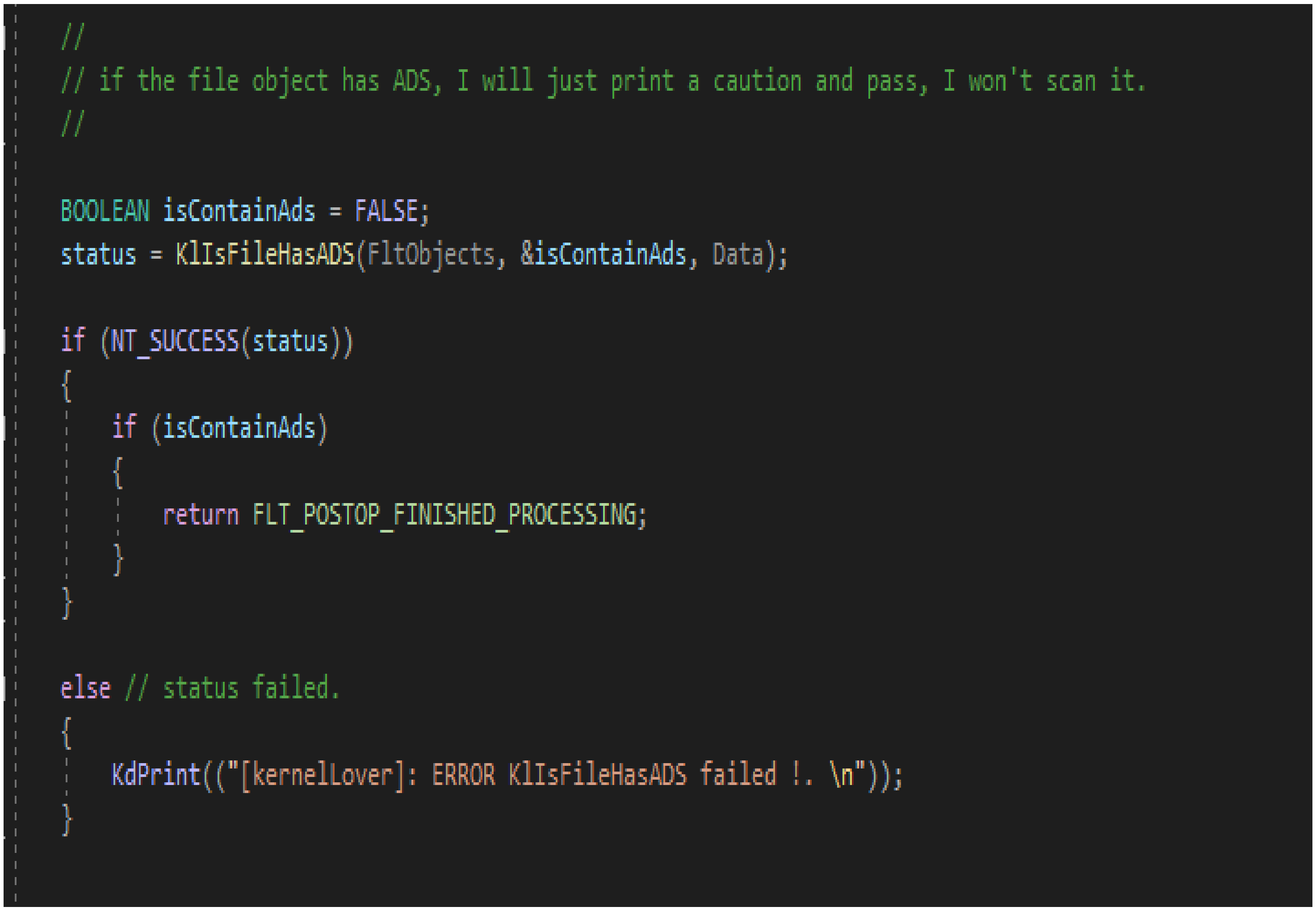


```
    if (KlIsFileInfected(streamContext->fileState))
    {
        KdPrint(("this file is infected. don't open it : %wZ\n", pFileNameInfo->Name));
        // do farther actions...
        FltCancelFileOpen(FltObjects->Instance, FltObjects->FileObject);

        Data->IoStatus.Status = STATUS_ACCESS_DENIED;
        Data->IoStatus.Information = 0;

        KdPrint(("[KL] SUCCESS: the file open is failed succesfully.\n"));
    }

    else
    {
        KdPrint(("[KL] the file is safe to open.\n"));
        Data->IoStatus.Status = STATUS_SUCCESS;
        Data->IoStatus.Information = 0;
    }

    if (pFileNameInfo != NULL)
        FltReleaseFileNameInformation(pFileNameInfo);

    goto CLEAN;

    //KdPrint(("Try to print the context ... \n"));

    //KdPrint(("Data is : %d \n", (ULONG_PTR)CompletionContext ));

CLEAN:
    FltReleaseContext(streamContext);
    return FLT_POSTOP_FINISHED_PROCESSING;

}
```

And in this function (which going to be called in the driver entry) I initialized a communication port between the mini filter driver and the user mode controller (or service), to communicate in a fast way.

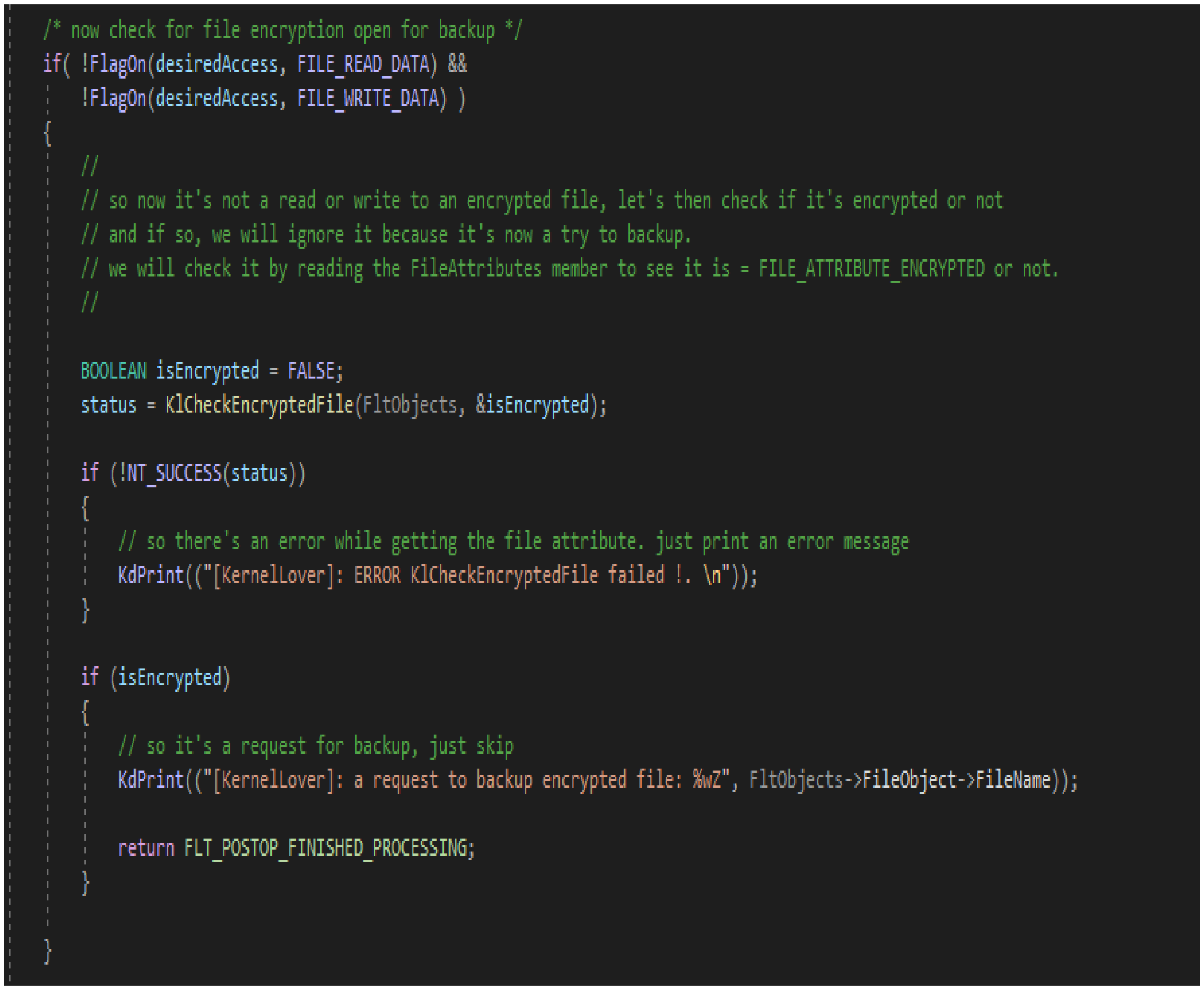


```
if (KlIsFileNeedScan(streamContext))
{
    BOOLEAN isInfected;
    SetFileScanning(streamContext->fileState); // set the scanning context

    POBJECT_NAME_INFORMATION objectNameInfo = (POBJECT_NAME_INFORMATION)ExAllocatePool2(POOL_FLAG_NON_PAGED, sizeof(OBJECT_NAME_INFORMATION), KL_MSDOS_FILE_NAME_TAG);

    // get the MSDOS FILE NAME
    status = IoQueryFileDosDeviceName(FltObjects->FileObject, &objectNameInfo);

    if (!NT_SUCCESS(status))
    {
        KdPrint(("[KL]ERROR: while calling IoQueryFileDosDeviceName.\n"));
    }
    else
    {
        KdPrint(("[KL]SUCCESS: the file name from IoQueryFileDosDeviceName is %wZ.\n", objectNameInfo->Name));
    }

    status = KlSendFileToEngine(objectNameInfo->Name, &isInfected);


    if (NT_SUCCESS(status))
    {
        if (isInfected)
        {
            SetFileInfected(streamContext->fileState);
            KdPrint(("[KL] this file name flaged as infected : %wZ\n", pFileNameInfo->Name));
        }

        else
        {
            SetFileCleared(streamContext->fileState);
            KdPrint(("[KL] this file name flaged as cleared: %wZ\n", pFileNameInfo->Name));
        }

        KdPrint(("[KL] NOTE: SCAN FINISHED.\n"));
    }
    else
        KdPrint(("[KL] ERROR: KlSendFileToEngine function failed.\n"));

    gNumofFileNeedScan++;

    if (objectNameInfo != NULL)
        ExFreePoolWithTag(objectNameInfo, KL_MSDOS_FILE_NAME_TAG);
}
else
    KdPrint(("[KL] NOTE: this file : %wZ DOES NOT need to be scanned.\n", FltObjects->FileObject->FileName));
```



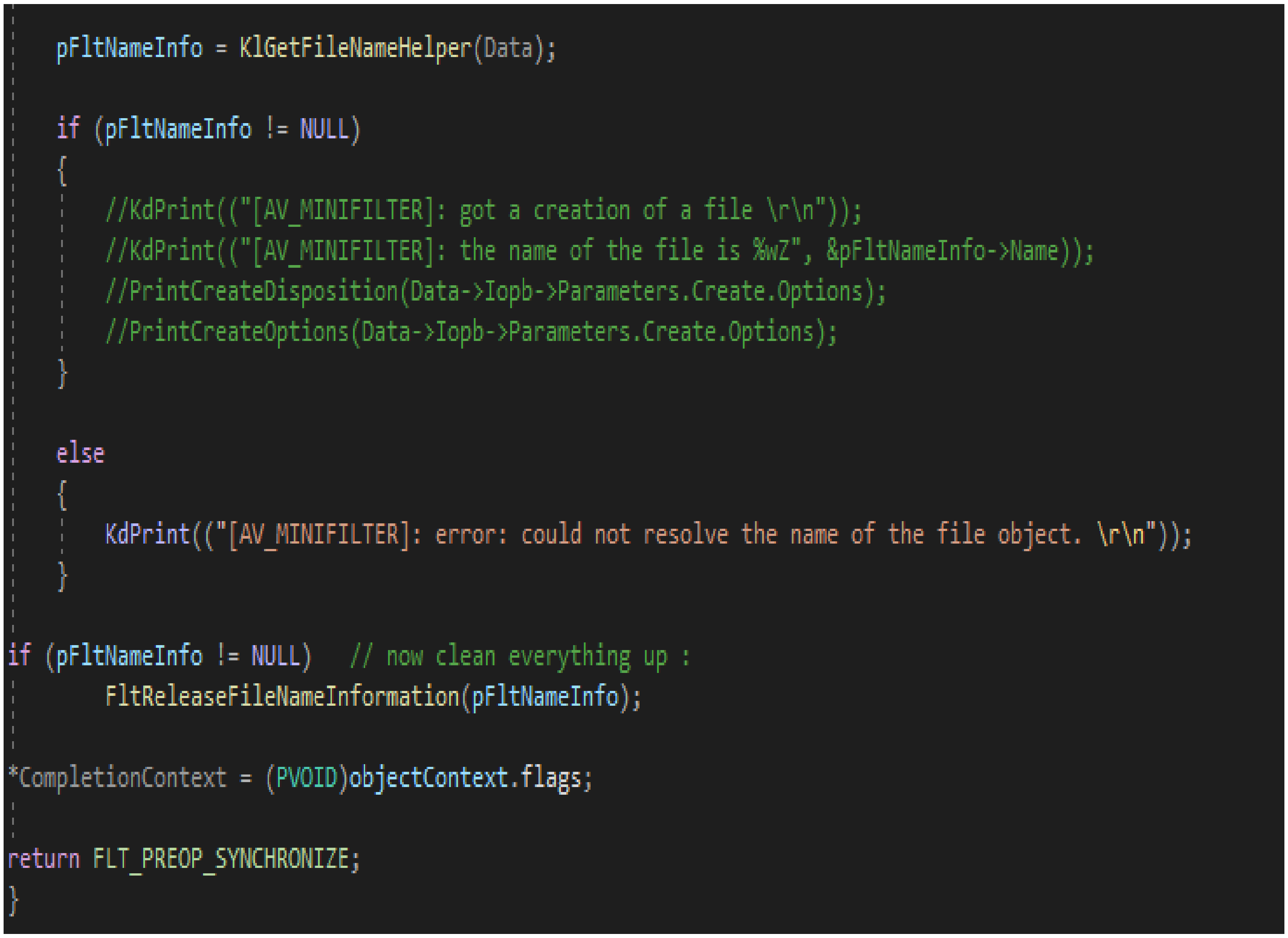
```
//
// if the file object has ADS, I will just print a caution and pass, I won't scan it.
//

BOOLEAN isContainAds = FALSE;
status = KlIsFileHasADS(FltObjects, &isContainAds, Data);

if (NT_SUCCESS(status))
{
    if (isContainAds)
    {
        return FLT_POSTOP_FINISHED_PROCESSING;
    }
}

else // status failed.
{
    KdPrint(("[kernelLover]: ERROR KlIsFileHasADS failed !. \n"));
}
```

In this function, we implemented how the driver will send a file path to the user mode controller to be scanned.



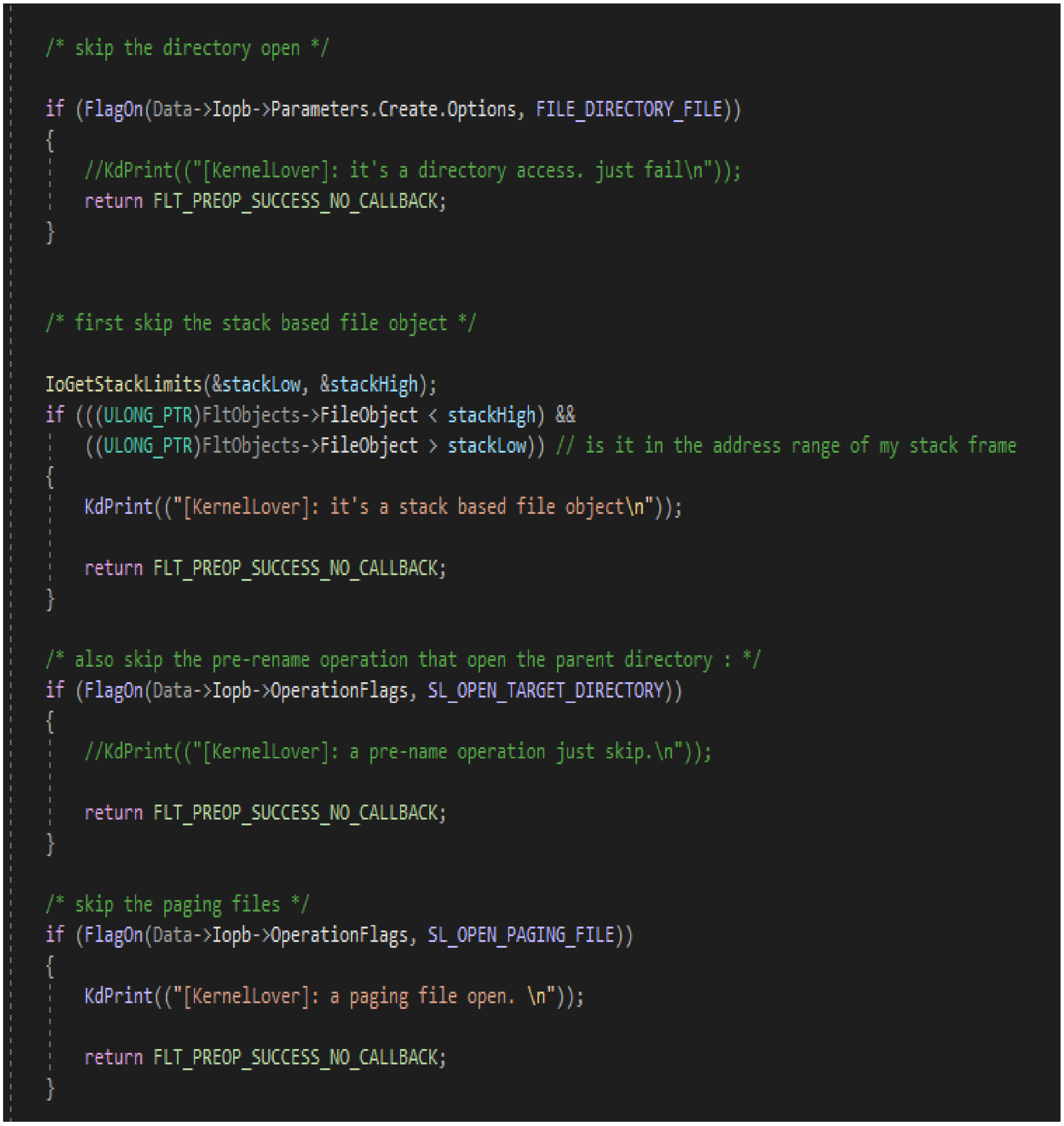
```
/* now check for file encryption open for backup */
if( !FlagOn(desiredAccess, FILE_READ_DATA) &&
    !FlagOn(desiredAccess, FILE_WRITE_DATA) )
{
    //
    // so now it's not a read or write to an encrypted file, let's then check if it's encrypted or not
    // and if so, we will ignore it because it's now a try to backup.
    // we will check it by reading the FileAttributes member to see it is = FILE_ATTRIBUTE_ENCRYPTED or not.
    //

    BOOLEAN isEncrypted = FALSE;
    status = KlCheckEncryptedFile(FltObjects, &isEncrypted);

    if (!NT_SUCCESS(status))
    {
        // so there's an error while getting the file attribute. just print an error message
        KdPrint(("[KernelLover]: ERROR KlCheckEncryptedFile failed !. \n"));
    }

    if (isEncrypted)
    {
        // so it's a request for backup, just skip
        KdPrint(("[KernelLover]: a request to backup encrypted file: %wZ", FltObjects->FileObject->FileName));

        return FLT_POSTOP_FINISHED_PROCESSING;
    }

}
```

And here, it's the implementation of the PreCreate Routine registered for IRP_MJ_CREATE. I skip many requests that's going to be useless for our case, like opening a directory or a dealing with prefetch files from kernel mode and so.



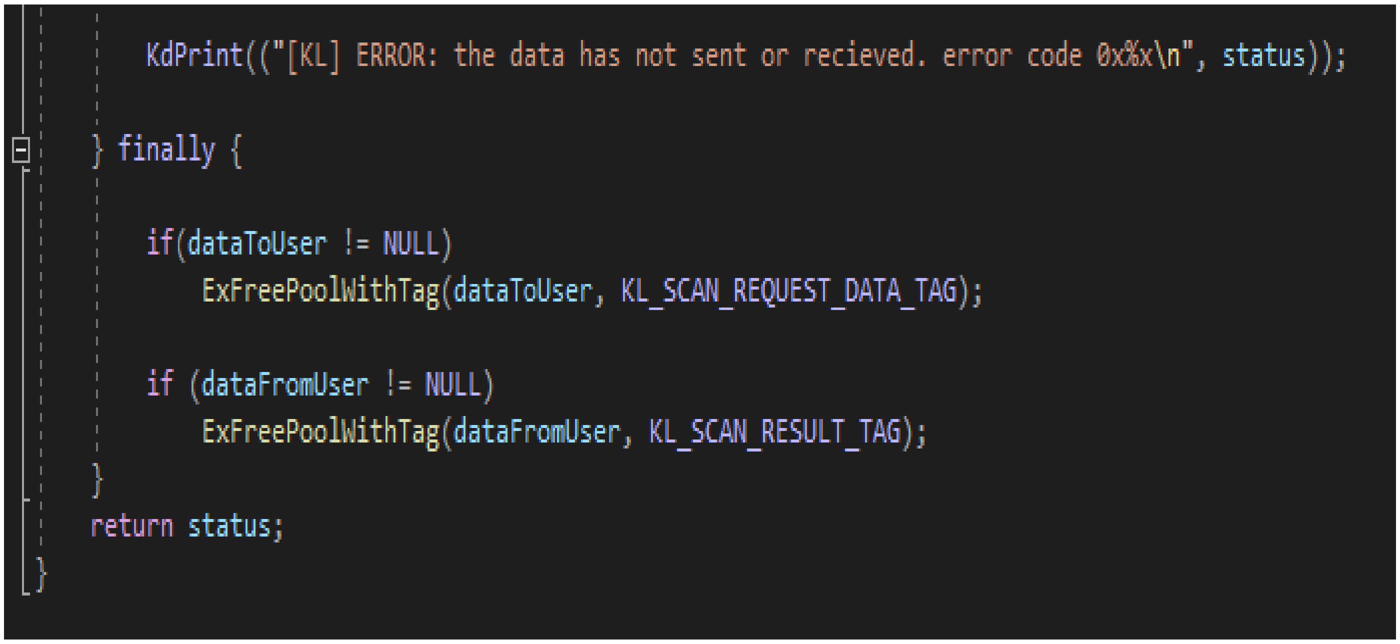
```
    pFltNameInfo = KlGetFileNameHelper(Data);

    if (pFltNameInfo != NULL)
    {
        //KdPrint(("[AV_MINIFILTER]: got a creation of a file \r\n"));
        //KdPrint(("[AV_MINIFILTER]: the name of the file is %wZ", &pFltNameInfo->Name));
        //PrintCreateDisposition(Data->Iopb->Parameters.Create.Options);
        //PrintCreateOptions(Data->Iopb->Parameters.Create.Options);
    }

    else
    {
        KdPrint(("[AV_MINIFILTER]: error: could not resolve the name of the file object. \r\n"));
    }

if (pFltNameInfo != NULL)   // now clean everything up :
        FltReleaseFileNameInformation(pFltNameInfo);

*CompletionContext = (PVOID)objectContext.flags;

return FLT_PREOP_SYNCHRONIZE;
}
```

And at the end when I find that this request is valid, I pass it to the post create routine.



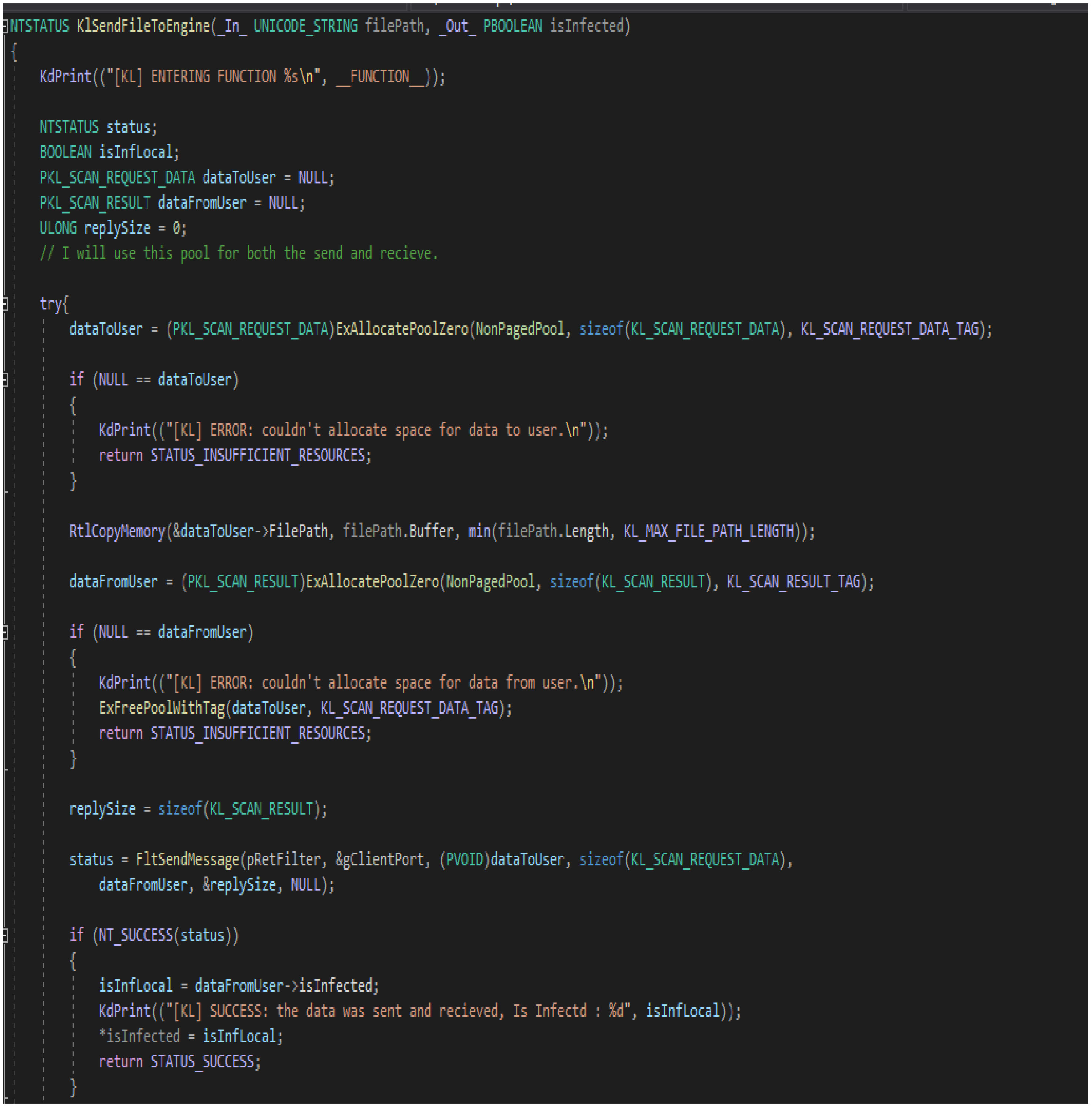
```
/* skip the directory open */

if (FlagOn(Data->Iopb->Parameters.Create.Options, FILE_DIRECTORY_FILE))
{
    //KdPrint(("[KernelLover]: it's a directory access. just fail\n"));
    return FLT_PREOP_SUCCESS_NO_CALLBACK;
}


/* first skip the stack based file object */

IoGetStackLimits(&stackLow, &stackHigh);
if (((ULONG_PTR)FltObjects->FileObject < stackHigh) &&
    ((ULONG_PTR)FltObjects->FileObject > stackLow)) // is it in the address range of my stack frame
{
    KdPrint(("[KernelLover]: it's a stack based file object\n"));

    return FLT_PREOP_SUCCESS_NO_CALLBACK;
}

/* also skip the pre-rename operation that open the parent directory : */
if (FlagOn(Data->Iopb->OperationFlags, SL_OPEN_TARGET_DIRECTORY))
{
    //KdPrint(("[KernelLover]: a pre-name operation just skip.\n"));

    return FLT_PREOP_SUCCESS_NO_CALLBACK;
}

/* skip the paging files */
if (FlagOn(Data->Iopb->OperationFlags, SL_OPEN_PAGING_FILE))
{
    KdPrint(("[KernelLover]: a paging file open. \n"));

    return FLT_PREOP_SUCCESS_NO_CALLBACK;
}
```

Also in the post create function, I do other checks like if the request is to read an encrypted file so skip that thing. also skip any i/o operation that's related to the prefetch file, create a stream context and so..



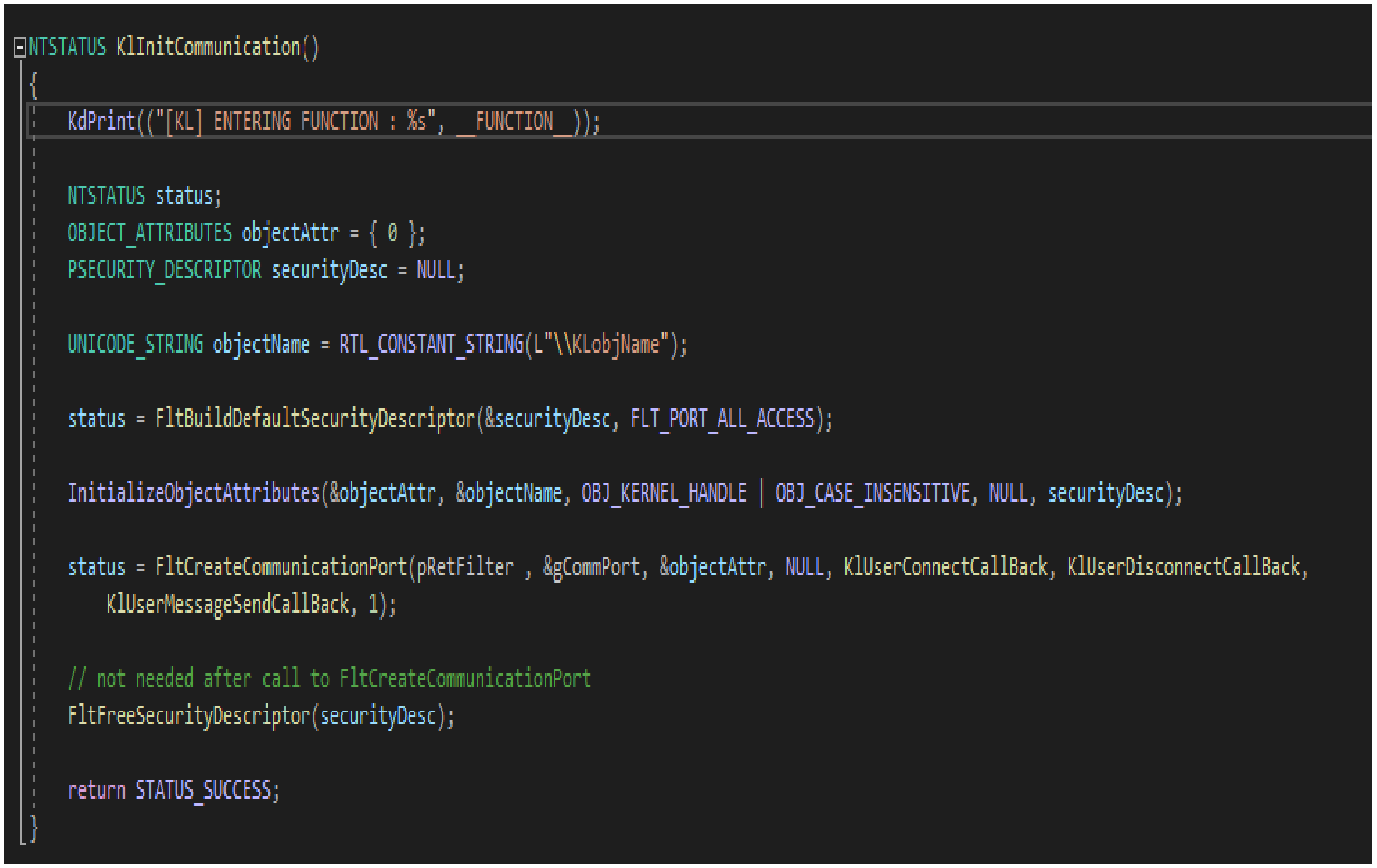
```
        KdPrint(("[KL] ERROR: the data has not sent or recieved. error code 0x%x\n", status));

    } finally {

        if(dataToUser != NULL)
            ExFreePoolWithTag(dataToUser, KL_SCAN_REQUEST_DATA_TAG);

        if (dataFromUser != NULL)
            ExFreePoolWithTag(dataFromUser, KL_SCAN_RESULT_TAG);
    }
    return status;
}
```

In this part I skip the ADS scanning, it's going to be available in the future.



```
NTSTATUS KlSendFileToEngine(_In_ UNICODE_STRING filePath, _Out_ PBOOLEAN isInfected)
{
    KdPrint(("[KL] ENTERING FUNCTION %s\n", __FUNCTION__));

    NTSTATUS status;
    BOOLEAN isInfLocal;
    PKL_SCAN_REQUEST_DATA dataToUser = NULL;
    PKL_SCAN_RESULT dataFromUser = NULL;
    ULONG replySize = 0;
    // I will use this pool for both the send and recieve.

    try{
        dataToUser = (PKL_SCAN_REQUEST_DATA)ExAllocatePoolZero(NonPagedPool, sizeof(KL_SCAN_REQUEST_DATA), KL_SCAN_REQUEST_DATA_TAG);

        if (NULL == dataToUser)
        {
            KdPrint(("[KL] ERROR: couldn't allocate space for data to user.\n"));
            return STATUS_INSUFFICIENT_RESOURCES;
        }

        RtlCopyMemory(&dataToUser->FilePath, filePath.Buffer, min(filePath.Length, KL_MAX_FILE_PATH_LENGTH));

        dataFromUser = (PKL_SCAN_RESULT)ExAllocatePoolZero(NonPagedPool, sizeof(KL_SCAN_RESULT), KL_SCAN_RESULT_TAG);

        if (NULL == dataFromUser)
        {
            KdPrint(("[KL] ERROR: couldn't allocate space for data from user.\n"));
            ExFreePoolWithTag(dataToUser, KL_SCAN_REQUEST_DATA_TAG);
            return STATUS_INSUFFICIENT_RESOURCES;
        }

        replySize = sizeof(KL_SCAN_RESULT);

        status = FltSendMessage(pRetFilter, &gClientPort, (PVOID)dataToUser, sizeof(KL_SCAN_REQUEST_DATA),
            dataFromUser, &replySize, NULL);

        if (NT_SUCCESS(status))
        {
            isInfLocal = dataFromUser->isInfected;
            KdPrint(("[KL] SUCCESS: the data was sent and recieved, Is Infectd : %d", isInfLocal));
            *isInfected = isInfLocal;
            return STATUS_SUCCESS;
        }
```

And in this function, I implemented the code that's going to decide whether the file need to be scanned or not, and the decision depends on the stream context to that file.



```
NTSTATUS KlInitCommunication()
{
    KdPrint(("[KL] ENTERING FUNCTION : %s", __FUNCTION__));

    NTSTATUS status;
    OBJECT_ATTRIBUTES objectAttr = { 0 };
    PSECURITY_DESCRIPTOR securityDesc = NULL;

    UNICODE_STRING objectName = RTL_CONSTANT_STRING(L"\\KLobjName");

    status = FltBuildDefaultSecurityDescriptor(&securityDesc, FLT_PORT_ALL_ACCESS);

    InitializeObjectAttributes(&objectAttr, &objectName, OBJ_KERNEL_HANDLE | OBJ_CASE_INSENSITIVE, NULL, securityDesc);

    status = FltCreateCommunicationPort(pRetFilter , &gCommPort, &objectAttr, NULL, KlUserConnectCallBack, KlUserDisconnectCallBack,
        KlUserMessageSendCallBack, 1);

    // not needed after call to FltCreateCommunicationPort
    FltFreeSecurityDescriptor(securityDesc);

    return STATUS_SUCCESS;
}
```

After that, the decision to do action if the file is malicious or not is implemented here, so after Identifying if the file is malicious, this code fails the request and return an access denied error. but if not it's not going to do anything.

5.1.3 Hypervisor Level: Hypervisor

At the Hypervisor Level, the system utilizes a Hypervisor to perform API hooking in kernel mode. The Hypervisor is responsible for intercepting system calls and logging them. This mechanism provides a robust layer of security, as the Hypervisor operates beneath the guest operating system, making it less detectable and more resistant to tampering by malicious software.

The Hypervisor hooks into the system calls by leveraging Extended Page Tables (EPT), which allows it to monitor memory accesses related to system calls. When a system call is invoked, the

EPT mechanism intercepts the memory access and redirects it to the Hypervisor. The Hypervisor then logs these API calls and sends the information to the FS Mini-Filter Driver.

5.1.4 Interaction Flow

The interaction flow begins with the user initiating a scan through the GUI Interface App. This app sends the scan request to the Engine Service, which starts processing the request. Meanwhile, the FS Mini-Filter Driver continuously monitors file operations and process creation at the kernel level. When it detects a relevant event, it communicates with the Hypervisor to log API calls related to the event.

The Hypervisor, having hooked into the system calls, logs the API calls and sends this information to the FS Mini-Filter Driver. The FS Mini-Filter Driver then forwards these logs to the Engine Service for analysis. The Engine Service examines the logs and the target object to determine if there is a threat. If the file is found to be malicious, the Engine Service can take appropriate actions, such as quarantining the file and notifying the user through the GUI Interface App.

Example Scenario

To illustrate the operation of this system, consider a scenario where a user downloads a new file and requests a scan:

1. The user opens the GUI Interface App and selects the newly downloaded file for scanning.
2. The GUI Interface App sends the scan request to the Engine Service.
3. The Engine Service acknowledges the request and begins processing.
4. Concurrently, the FS Mini-Filter Driver, which is monitoring all file operations, detects the creation of the new file.
5. The FS Mini-Filter Driver requests the Hypervisor to log API calls related to the file creation.
6. The Hypervisor intercepts these system calls, logs them, and sends the logs to the FS Mini-Filter Driver.
7. The FS Mini-Filter Driver forwards the logs to the Engine Service.
8. The Engine Service analyzes the file and the logs to determine if the file is malicious.

9. If the file is safe, the Engine Service sends a clean result back to the GUI Interface App, and the user is informed.
10. If the file is malicious, the Engine Service takes action, such as quarantining the file and alerting the user through the GUI Interface App.

This design integrates components operating at different levels to provide a robust security monitoring and scanning system. By leveraging user-mode applications, kernel-mode drivers, and hypervisor-level hooks, the system can effectively detect and respond to potential threats. The use of Extended Page Tables (EPT) for system call hooking ensures efficient and less detectable monitoring, enhancing the overall security posture of the system.

5.2 Advantages of Hypervisor-Based AV

In today's digital age, protecting our computers from viruses and malware is crucial. Traditional anti-virus solutions have been around for a long time, but a newer, more advanced method is gaining popularity: hypervisor-based anti-virus. This approach leverages the power of hypervisors to enhance security and performance. Let's explore the advantages of this innovative technology.

5.2.1 How Hypervisor-Based Anti-Virus Works

A hypervisor is a software layer that allows multiple operating systems to run on a single physical machine. It creates and manages virtual machines (VMs) by abstracting the hardware. Hypervisor-based anti-virus solutions integrate security features directly into this layer, offering a unique way to detect and prevent threats. By operating outside the operating system, these solutions can monitor and manage the system more effectively, without the limitations faced by traditional anti-virus software.

5.2.2 Key Advantages

Enhanced Security

One of the main benefits of hypervisor-based anti-virus is enhanced security. Traditional anti-virus programs operate within the same layer as the operating system, making them vulnerable to advanced malware that can disable or bypass them. In contrast, hypervisor-based solutions operate at a lower layer, outside the reach of most malware. This makes it much harder for malicious software to evade detection and removal.

Deeper Inspection:

Hypervisor-based anti-virus can perform deeper inspections of system activities because it has a broader view of the system's operations. This allows for more thorough detection of hidden threats.

1. **Rootkit Detection:** Rootkits are particularly dangerous types of malware that hide their presence from traditional anti-virus software. Hypervisor-based solutions can detect and neutralize rootkits by observing their behavior from a privileged layer.
2. **Tamper-Resistant:** Since hypervisor-based anti-virus runs below the operating system, it is less susceptible to tampering by malware that targets traditional anti-virus programs.

Improved Performance

Performance is another critical advantage. Traditional anti-virus programs can slow down your computer, especially during system scans or updates. Hypervisor-based solutions are designed to minimize this impact. By operating at the hypervisor level, they can efficiently monitor and manage resources, ensuring that your computer runs smoothly while staying protected.

1. **Resource Optimization:** Hypervisor-based anti-virus can optimize the allocation of system resources, ensuring that security processes do not interfere with the normal operation of the system.
2. **Efficient Scanning:** These solutions can perform scanning and threat detection more efficiently by leveraging the hypervisor's ability to manage multiple virtual machines simultaneously.
3. **Low Overhead:** By integrating directly with the hypervisor, these solutions can reduce the overhead typically associated with traditional anti-virus software, resulting in better overall system performance.

Better System Integration

Hypervisor-based anti-virus solutions offer better integration with your system. They are designed to work seamlessly with various operating systems and applications. This means they can provide consistent protection across different environments without compatibility issues. Additionally, they are easier to deploy and manage, making them ideal for both personal and enterprise use.

1. **Cross-Platform Compatibility:** Hypervisor-based solutions can protect multiple operating systems running on the same hardware, providing a unified security solution for diverse environments.

2. **Centralized Management:** These solutions often come with centralized management tools that make it easier to deploy, configure, and monitor security across multiple systems.

3. **Scalability:** Hypervisor-based anti-virus can easily scale to accommodate growing environments, making it suitable for both small and large organizations.

Real-time Threat Detection

Real-time threat detection is crucial in preventing malware from causing harm. Hypervisor-based solutions excel in this area by continuously monitoring system activity. They can detect suspicious behavior and potential threats as they happen, allowing for immediate response and mitigation. This proactive approach ensures that your system remains secure at all times.

1. **Continuous Monitoring:** Hypervisor-based anti-virus continuously monitors all virtual machines and the hypervisor itself, providing real-time detection of threats.

2. **Behavior Analysis:** These solutions can analyze the behavior of applications and processes in real-time, identifying anomalies that may indicate the presence of malware.

3. **Instant Response:** By detecting threats in real-time, hypervisor-based anti-virus can respond immediately to neutralize them, reducing the risk of damage to the system.

Isolation of Malicious Software

Isolation is a powerful technique used by hypervisor-based anti-virus solutions to contain and neutralize threats. By creating isolated environments (virtual machines), they can safely execute and analyze suspicious files without risking the host system. This means even if a piece of malware manages to get through, it can be confined and dealt with effectively without affecting your main system.

1. **Virtual Machine Isolation:** Hypervisor-based solutions can isolate each virtual machine, preventing malware from spreading across the system.

2. **Safe Execution:** Suspicious files can be executed in a controlled, isolated environment where their behavior can be analyzed without risk.

3. **Containment:** If a threat is detected, it can be contained within a virtual machine, preventing it from affecting other parts of the system.

Reduced Impact on Host System

Finally, hypervisor-based anti-virus solutions have a reduced impact on the host system. They consume fewer resources compared to traditional anti-virus programs, which often require significant CPU and memory usage. This results in a smoother and more responsive computing experience, with less interference in your day-to-day activities.

1. **Lower Resource Consumption:** Hypervisor-based anti-virus uses fewer system resources, leaving more available for other applications and processes.
2. **Minimal Interference:** These solutions operate quietly in the background, minimizing interruptions to the user experience.
3. **Enhanced User Experience:** With less impact on system performance, users can enjoy a more responsive and efficient computing environment.

5.3 EPT system call hooking

In modern computing, system calls are a fundamental mechanism by which applications interact with the operating system. System call hooking is a technique used to intercept and potentially modify these calls, often for purposes such as security monitoring, debugging, and system enhancement. One advanced method for implementing system call hooking involves the use of Extended Page Tables (EPT). This section will explore what EPT is, how it can be used for system call hooking, and the benefits and applications of this approach.

5.3.1 What is EPT?

Extended Page Tables (EPT) is a hardware-assisted virtualization technology provided by Intel's VT-x. It is used to manage memory virtualization, allowing a hypervisor to efficiently map guest physical addresses to host physical addresses. EPT provides an additional layer of page tables, which enhances performance and simplifies memory management in virtualized environments.

5.3.2 Detailed Explanation of EPT

1. **Memory Virtualization**:
    - Traditional memory management involves translating virtual addresses to physical addresses using page tables.
    - In virtualized environments, a guest operating system manages its own virtual-to-physical address mappings. However, these "guest physical addresses" need further translation to the actual physical addresses on the host machine. EPT facilitates this by providing a second layer of translation.

2. **EPT Structure**:

    - EPT consists of multiple levels of page tables: EPTP (EPT Pointer), PML4, PDPT, PD, and PT.
    - Each level of the table holds entries that point to the next level, ultimately mapping guest physical addresses to host physical addresses.

3. **Benefits of EPT**:

    - **Performance**: EPT reduces the overhead associated with traditional shadow page tables, leading to faster memory operations in virtualized environments.
    - **Flexibility**: It allows the hypervisor to control memory access with fine granularity, providing capabilities such as read/write/execute permissions on a per-page basis.

5.3.3 How System Call Hooking Works

System call hooking involves intercepting the execution flow when a system call is made, allowing the hook to perform additional actions or modify the behavior of the system call. Traditional methods of hooking might involve modifying the system call table or using inline hooks, but these techniques can be detected and bypassed by sophisticated malware. EPT provides a more robust and less detectable method for hooking system calls.

5.3.4 Traditional Hooking Methods

1. **System Call Table Hooking**:

    - The system call table (or syscall table) contains pointers to the system call handlers.
    - By replacing an entry in this table, one can redirect the execution to custom code.

    Drawback: This method is easily detectable by integrity checks and can be tampered with by advanced malware.

2. **Inline Hooking**:

    Inline hooks modify the first few instructions of a system call handler to jump to custom code.

    Drawback: This method is also detectable and can be unstable if the instructions being modified are critical.

5.3.5 EPT System Call Hooking Mechanism

1. **Memory Virtualization with EPT**:
   - EPT enables the hypervisor to control memory access more granularly.
   - By leveraging EPT, the hypervisor can monitor and intercept memory accesses related to system calls.
2. **Setting Up Hooks**:
   - The hypervisor sets up EPT entries to monitor specific memory regions where system call code resides.
   - When a system call is invoked, the EPT mechanism intercepts the memory access and redirects it to the hypervisor.

5.3.5.1 Detailed Hooking Process

1. **Identify System Call Code**:
   - The hypervisor first identifies the memory regions corresponding to system call handlers in the guest OS.
   - This can be done by inspecting the guest OS's system call table or specific kernel code segments.
2. **Configure EPT Entries**:
   - The hypervisor configures EPT entries to trigger VM exits (interceptions) when certain conditions are met, such as execution of code in specific memory regions.
   - EPT entries are set with special permissions or access flags to monitor read/write/execute operations.
3. **Intercepting System Calls**:
   - When the guest OS invokes a system call, the EPT triggers a VM exit, transferring control to the hypervisor.
   - The hypervisor then inspects the context of the system call, including parameters and the intended system call number.

4. **Handling Intercepts**:

    - The hypervisor processes the intercepted call, allowing it to inspect or modify the parameters and behavior of the system call.
    - After handling the intercept, the hypervisor can decide to pass control back to the original system call handler or modify the execution flow.

5.3.6 Advantages of EPT System Call Hooking

1. **Invisibility to Guest OS**:

    - EPT-based hooks are implemented at the hypervisor level, making them invisible to the guest operating system.
    - This reduces the risk of detection and tampering by malicious software running in the guest OS.

2. **Improved Security**:

    - By operating outside the guest OS, EPT hooks can provide more robust security monitoring and enforcement.
    - This approach is effective against rootkits and other advanced threats that attempt to subvert traditional hooking mechanisms.

5.3.6.1 Detailed Security Benefits

1. **Deeper Inspection**:

    - EPT-based solutions can perform deeper inspections of system activities because they have a broader view of the system's operations.
    - This allows for more thorough detection of hidden threats and anomalies.

2. **Rootkit Detection**:

Rootkits are particularly dangerous types of malware that hide their presence from traditional anti-virus software. Hypervisor-based solutions can detect and neutralize rootkits by observing their behavior from a privileged layer.

3. **Tamper-Resistant**:

Since EPT-based anti-virus runs below the operating system, it is less susceptible to tampering by malware that targets traditional anti-virus programs.

5.3.7 Improved Performance

1. **Resource Optimization**:

Hypervisor-based anti-virus can optimize the allocation of system resources, ensuring that security processes do not interfere with the normal operation of the system.

2. **Efficient Scanning**:

These solutions can perform scanning and threat detection more efficiently by leveraging the hypervisor's ability to manage multiple virtual machines simultaneously.

3. **Low Overhead**:

By integrating directly with the hypervisor, these solutions can reduce the overhead typically associated with traditional anti-virus software, resulting in better overall system performance.

5.3.8 Better System Integration

1. **Cross-Platform Compatibility**:

Hypervisor-based solutions can protect multiple operating systems running on the same hardware, providing a unified security solution for diverse environments.

2. **Centralized Management**:

These solutions often come with centralized management tools that make it easier to deploy, configure, and monitor security across multiple systems.

3. **Scalability**:

Hypervisor-based anti-virus can easily scale to accommodate growing environments, making it suitable for both small and large organizations.

5.3.9 Real-time Threat Detection

1. **Continuous Monitoring**:

Hypervisor-based anti-virus continuously monitors all virtual machines and the hypervisor itself, providing real-time detection of threats.

2. **Behavior Analysis**:

These solutions can analyze the behavior of applications and processes in real-time, identifying anomalies that may indicate the presence of malware.

3. **Instant Response**:

By detecting threats in real-time, hypervisor-based anti-virus can respond immediately to neutralize them, reducing the risk of damage to the system.

5.3.10 Isolation of Malicious Software

1. **Virtual Machine Isolation**:

Hypervisor-based solutions can isolate each virtual machine, preventing malware from spreading across the system.

2. **Safe Execution**:

Suspicious files can be executed in a controlled, isolated environment where their behavior can be analyzed without risk.

3. **Containment**:

If a threat is detected, it can be contained within a virtual machine, preventing it from affecting other parts of the system.

5.3.11 Reduced Impact on Host System

1. **Lower Resource Consumption**:

Hypervisor-based anti-virus uses fewer system resources, leaving more available for other applications and processes.

2. **Minimal Interference**:

These solutions operate quietly in the background, minimizing interruptions to the user experience.

3. **Enhanced User Experience**:

With less impact on system performance, users can enjoy a more responsive and efficient computing environment.

5.3.12 Applications of EPT System Call Hooking

1. **Security Monitoring**:
    - EPT hooks can be used to monitor and log system calls for security analysis and intrusion detection.
    - This helps in identifying and mitigating malicious activities that attempt to exploit system calls.
2. **System Hardening**:
    - By intercepting and analyzing system calls, EPT hooks can enforce security policies and prevent unauthorized actions.
    - This enhances the overall security posture of the system.
3. **Debugging and Diagnostics**:
    - Developers can use EPT system call hooking to trace and diagnose issues in software applications and the operating system.
    - It provides a powerful tool for understanding system behavior and identifying bugs.
4. **Virtual Machine Introspection**:
    - EPT-based hooks enable advanced virtual machine introspection (VMI) techniques, allowing security tools to analyze the state of virtual machines without interfering with their normal operation.
    - This is particularly useful in cloud environments where multiple virtual machines run on the same physical hardware.

5.4 Challenges

Hypervisor-based anti-virus solutions represent a significant advancement in cybersecurity, offering the ability to monitor and protect systems at a deeper level than traditional methods. However, the implementation of such solutions involves navigating several complex challenges, including ensuring compatibility with different platforms, handling unexpected conditions to avoid system crashes, and maintaining high performance. This discussion explores these challenges within the context of hypervisor-based anti-virus solutions.

5.4.1 Compatibility with Different Platforms

Ensuring compatibility with various operating systems is a crucial challenge for hypervisor-based anti-virus solutions. Each platform—Linux, macOS, and Windows—has distinct characteristics and architectural differences that must be addressed to implement a universal solution effectively.

5.4.1.1 Linux Compatibility

Linux, known for its flexibility and open-source nature, offers both opportunities and challenges for hypervisor-based anti-virus solutions. The diversity of Linux distributions, each with its unique kernel versions and configurations, requires the solution to be highly adaptable. The hypervisor must support a wide range of kernel versions and be compatible with various security modules like SELinux or AppArmor. Integration with these security modules is essential to avoid conflicts and ensure comprehensive system protection. Additionally, leveraging open-source tools and libraries commonly used in Linux environments can enhance compatibility and performance.

5.4.1.2 macOS Compatibility

macOS presents different challenges due to its unique architecture and stringent security measures. The introduction of System Integrity Protection (SIP) in macOS restricts the operations of kernel extensions, which can limit the functionality of a hypervisor-based solution. The solution must be designed to comply with SIP while still achieving deep system monitoring capabilities. Utilizing specific macOS APIs and frameworks, such as the Endpoint Security Framework, can facilitate better integration and performance. Furthermore, the solution must maintain the high user experience standards expected by macOS users, ensuring it does not degrade system performance or disrupt user interactions.

5.4.1.3 Windows Compatibility

Windows, the most widely used operating system in enterprise environments, requires careful consideration due to its complex architecture and numerous versions. The hypervisor-based solution must support a broad range of Windows versions, each with specific features and security mechanisms. Developing kernel-level drivers for Windows involves ensuring these drivers are signed and certified by Microsoft, which is crucial for maintaining system stability and security. Additionally, integrating with Windows security tools, such as Windows Defender and System Guard, is essential for providing comprehensive protection. The solution should complement these tools, enhancing overall system security without redundancy or conflict.

5.4.2 Handling Unexpected Conditions to Avoid System Crashes

A critical challenge for hypervisor-based anti-virus solutions is ensuring they can handle unexpected conditions without causing system crashes, such as the Blue Screen of Death (BSOD) in Windows or kernel panics in Linux and macOS.

5.4.2.1 Robust Error Handling

Implementing thorough error handling routines is essential for managing unexpected conditions effectively. Proper input validation and sanitation can prevent common issues like buffer overflows and null pointer dereferences, which can lead to system crashes. Exception handling mechanisms, both at the user and kernel levels, can catch and manage exceptions to prevent minor issues from escalating into critical failures. For instance, using try-catch blocks in the code and appropriate kernel exception handling mechanisms can ensure the system remains stable even when errors occur.

5.4.2.2 Fail-Safe Mechanisms

Designing fail-safe mechanisms is crucial for maintaining system stability during unexpected conditions. Implementing fallback procedures that activate when the primary monitoring process fails can help ensure the system continues to operate without crashing. For example, if a hook fails, the system can revert to a safe default state, allowing operations to continue while alerting administrators of the issue. Graceful degradation, where the monitoring engine reduces its functionality instead of crashing, can also help maintain minimal but crucial monitoring capabilities while addressing the underlying problem.

5.4.2.3 Testing and Validation

Extensive testing and validation are essential to ensure the hypervisor-based solution can handle unexpected conditions effectively. Conducting stress tests under high loads and various operational conditions helps identify and address potential stability issues. Simulating real-world scenarios, including different types of attacks and system configurations, ensures the solution can manage diverse situations without failing. This thorough testing process is crucial for ensuring the reliability and robustness of the hypervisor-based solution.

5.4.3 Maintaining High Performance

Balancing robust security with high performance is a significant challenge for hypervisor-based anti-virus solutions. Enhancing performance involves optimizing resource utilization, reducing overhead, and ensuring seamless operation without compromising security.

5.4.3.1 Resource Optimization

Efficiently managing system resources is vital for maintaining performance while conducting thorough monitoring. One approach is selective monitoring, where the hypervisor focuses on high-risk processes and critical system activities, significantly reducing resource consumption. Caching mechanisms can also be implemented to store frequently accessed data, reducing redundant processing. Additionally, batching operations to process multiple events simultaneously can improve efficiency and reduce the overall resource load on the system.

5.4.3.2 Minimizing Overhead

Reducing the processing overhead associated with monitoring activities is essential to ensure minimal impact on system performance. Low-level optimization, such as writing performance-critical components in languages like C or assembly and optimizing code paths for speed, can minimize the overhead introduced by monitoring. Asynchronous processing techniques can also be employed, allowing the hypervisor to handle multiple tasks concurrently without blocking the main system operations, thus enhancing performance.

5.4.3.3 Leveraging Hardware Acceleration

Utilizing hardware capabilities can significantly enhance the performance of hypervisor-based anti-virus solutions. Hardware-assisted virtualization technologies, such as Intel VT-x and AMD-V, provide support for efficient and low-overhead kernel-level monitoring. These technologies enable the hypervisor to operate with minimal performance impact while still providing comprehensive system analysis. Additionally, GPU acceleration can be utilized for data-intensive tasks, offloading work from the CPU and speeding up analysis processes, further improving the overall performance of the solution.

Hypervisor-based anti-virus solutions offer advanced capabilities for system protection but also present several complex challenges. Ensuring compatibility with different platforms requires addressing the unique characteristics of Linux, macOS, and Windows. Robust error handling and fail-safe mechanisms are essential for managing unexpected conditions and avoiding system crashes. Enhancing performance involves optimizing resource utilization, minimizing processing overhead, and leveraging hardware acceleration to maintain robust security without compromising system efficiency. Successfully navigating these challenges is crucial for developing effective and reliable hypervisor-based anti-virus solutions that can protect against sophisticated and rapidly evolving threats.

5.5 Engine Service: Analyzing using YARA rules

Enhancing Virtualization Security: Leveraging YARA Rules in Hypervisor Antivirus Systems

Virtualization technology has revolutionized IT infrastructures, enabling organizations to optimize resource utilization, enhance scalability, and streamline management. However, with these benefits come unique cyber security challenges, particularly in ensuring the protection of virtualized environments against evolving malware threats. Traditional antivirus solutions, which operate within individual virtual machines (VMs), may struggle to provide comprehensive security across dynamic and interconnected virtual infrastructures. To address these challenges, hypervisor antivirus systems leverage advanced techniques such as YARA rules, enhancing their ability to detect and mitigate sophisticated malware threats at the hypervisor level.

5.5.1 Understanding Hypervisor Antivirus Systems

Hypervisor antivirus systems represent the next evolution in virtualization security. Unlike traditional endpoint antivirus solutions that operate within guest operating systems, hypervisor antivirus operates directly within the hypervisor layer. This strategic positioning allows it to monitor and protect multiple VMs simultaneously, providing centralized security management and reducing performance overhead on individual VMs. By intercepting and inspecting system calls, network traffic, and storage activities at the hypervisor level, these solutions offer a holistic approach to virtualization security.

5.5.2 The Role of YARA Rules in Malware Detection

At the heart of effective malware detection within hypervisor antivirus systems lies YARA (Yet Another Recursive acronym) rules. YARA is an open-source tool designed for identifying and classifying malware based on textual or binary patterns. YARA rules are defined using a flexible and expressive syntax that allows cyber security professionals to specify conditions and patterns indicative of malicious behavior. These rules can range from simple string matches to complex logical expressions and are highly adaptable to detect both known and emerging threats.

5.5.3 Advantages of Integrating YARA Rules

Integrating YARA rules into the engine service of hypervisor antivirus systems offers several strategic advantages:

1. **Comprehensive Threat Detection**: YARA rules enable hypervisor antivirus systems to detect a wide range of malware variants, including polymorphic and obfuscated threats that may evade traditional signature-based detection methods.
2. **Behavioral Analysis**: By defining rules that detect specific behavioral patterns or indicators of compromise (IOCs), organizations can enhance their ability to identify malicious activities across virtualized environments.
3. **Real-time Updates**: YARA rules can be updated in real-time to respond to emerging threats or newly identified malware strains, ensuring that the hypervisor antivirus system remains effective against evolving cybersecurity threats.
4. **Reduced Performance Impact**: Operating at the hypervisor level minimizes performance overhead on individual VMs, optimizing resource utilization and maintaining operational efficiency across the virtualized infrastructure.

5.5.4 Implementing YARA Rules: Best Practices

Effective implementation of YARA rules within a hypervisor antivirus system involves the following best practices:

- **Rule Management**: Establishing a structured process for creating, updating, and managing YARA rules based on threat intelligence feeds, security research, and incident response findings.
- **Performance Optimization**: Leveraging techniques such as distributed scanning, resource scheduling, and intelligent caching to optimize scanning processes and minimize impact on hypervisor performance.
- **Integration with Security Ecosystem**: Seamless integration with existing security information and event management (SIEM) systems, threat intelligence platforms, and security orchestration tools enhances visibility, facilitates automated response actions, and streamlines incident management workflows.
- **Compliance and Reporting**: Utilizing YARA rules to enhance threat detection capabilities aids in demonstrating compliance with industry regulations and security standards. Comprehensive reporting capabilities provide stakeholders with visibility into threat detection, mitigation actions, and overall security posture.

In conclusion, integrating YARA rules into the engine service of hypervisor antivirus systems represents a proactive approach to enhancing virtualization security. By leveraging YARA's robust pattern matching capabilities, organizations can strengthen their defenses against complex and evolving malware threats within dynamic virtualized environments. This approach not only improves detection efficacy but also enhances operational efficiency, reduces cyber security risk, and ensures comprehensive protection across interconnected VMs. As virtualization continues to play a pivotal role in modern IT infrastructures, the adoption of advanced security measures such as YARA rules within hypervisor antivirus systems becomes increasingly essential to safeguarding critical assets and maintaining business continuity.

6

Future Work

6.1 Generic AV Engine

>> Despite the kernel components of the system provide the generic anti-virus design, We seek to build generic engine for the anti-virus.

In today's digital landscape, where cyber threats are constantly evolving, anti-virus (AV) engines are crucial for protecting computers and networks. A generic anti-virus engine is designed to detect, prevent, and remove various types of malicious software, including viruses, worms, trojans, ransomware, and spyware. These engines are the backbone of modern cyber security defenses, serving as the primary line of defense against malicious software that can compromise data integrity, confidentiality, and availability.

Anti-virus engines operate through several key functions. Signature-based detection relies on a database of known malware signatures, which the engine uses to identify and flag malicious files. Heuristic analysis helps detect new and unknown malware by analyzing the behavior and characteristics of files and programs, identifying suspicious patterns even without a specific signature. Behavioral analysis goes further by monitoring real-time program behavior to detect actions typically associated with malware, such as unauthorized system file access or unusual network activity. Sandboxing involves executing suspicious files in a controlled, isolated environment to observe their behavior without risking the actual system. If a file exhibits malicious behavior in the sandbox, it is blocked from executing on the real system. Modern anti-virus engines also leverage cloud technology to access a vast database of threat intelligence in real-time, ensuring up-to-date protection and improved performance by offloading processing from the local device.

The importance of generic anti-virus engines is clear in the context of contemporary cyber security. They provide comprehensive protection against a wide range of threats, offering real-time defense that prevents malware from executing and causing harm. This protection helps maintain data integrity and security by preventing unauthorized access and data breaches, which is essential for individuals and organizations handling sensitive information. The presence of a robust anti-virus engine also boosts user confidence in their digital environment, allowing them to operate without constant fear of cyber threats.

The generic anti-virus engine is a cornerstone of modern cyber security, designed to detect, prevent, and eliminate a wide array of malicious software. Through a combination of signature-based detection, heuristic and behavioral analysis, sandboxing, and cloud-based protection, these engines provide comprehensive and real-time defense against evolving threats. As cyber threats continue to grow in sophistication and frequency, the role of anti-virus engines in safeguarding digital environments remains indispensable.

6.2 ADS Scanning

Alternate Data Streams (ADS) are a feature of the NTFS (New Technology File System) used by Windows that allows files to contain multiple streams of data. This capability was initially introduced to provide compatibility with the Hierarchical File System (HFS) used by Macintosh computers, which also supported multiple data streams. ADS enables users to attach hidden, secondary data streams to a file without changing its primary data stream or visible file size.

6.2.1 How ADS Works

In an NTFS file system, every file and directory is an object, and each object can have multiple data streams. The primary data stream is the main content of the file that is typically visible to users and applications. ADS allows additional, hidden streams to be attached to this primary stream. These alternate streams can store various types of data, such as metadata, additional file content, or even executable code.

6.2.2 Uses of ADS

1. **Compatibility and Metadata**: ADS is used to store metadata or additional file attributes without altering the main file content. For example, some antivirus software uses ADS to store information about scanned files.

2. **Application Data**: Some applications may use ADS to store additional data related to a file, such as thumbnails, user comments, or settings.

3. **Digital Forensics**: Forensic investigators may check for ADS as they can contain hidden data that is crucial for an investigation.

6.2.3 Security Risks of ADS

While ADS has legitimate uses, it also poses significant security risks. Malicious actors can exploit ADS to hide malicious code or data, making it harder to detect and remove. Because alternate streams do not appear in standard file listings or affect the reported file size, they can be used to conceal malware or sensitive information.

- **Malware Concealment:** Malware can use ADS to hide its presence on a system. By storing malicious code in an alternate stream, the malware can avoid detection by traditional antivirus software that does not scan ADS.
- **Data Exfiltration:** ADS can be used to smuggle sensitive data out of a system. An attacker can hide sensitive information in an alternate stream and transfer the file without raising suspicion.

Alternate Data Streams (ADS) are a powerful feature of the NTFS file system that allows files to have multiple data streams. While ADS can be used for legitimate purposes such as storing metadata or application data, they also pose significant security risks due to their ability to hide data and malicious code. Understanding how ADS works and using appropriate tools to detect and manage them is crucial for maintaining system security.

>> In the future, our anti-virus will be able to scan the ADS.

A hypervisor-based anti-virus solution operates at a lower level than traditional anti-virus software, providing several advantages, including enhanced security and the ability to monitor and control system activities more effectively. Scanning Alternate Data Streams (ADS) is crucial because they can be used to hide malicious code and evade detection. Here’s how a hypervisor-based anti-virus can be designed to scan ADS:

Understanding the Role of the Hypervisor

A hypervisor operates at the hardware level, below the operating system, creating and managing virtual machines. This low-level operation allows the hypervisor to monitor and control system calls, file operations, and other activities, providing a robust platform for anti-virus operations. The hypervisor can intercept and analyze file operations, including those involving ADS.

Steps for Scanning ADS with a Hypervisor-Based Anti-Virus

1. **Interception of File Operations**: The hypervisor can intercept all file operations, including the creation, modification, and reading of files. By hooking into these operations, the hypervisor can detect when a file with an ADS is accessed or modified.

2. **Accessing ADS Information**: When a file operation is intercepted, the hypervisor can use NTFS-specific APIs to query and access any ADS associated with the file. This can be done by leveraging low-level file system APIs that are capable of enumerating all data streams attached to a file.

3. **Scanning ADS for Malicious Content**: Once the ADS data is accessed, it can be passed to the anti-virus scanning engine, just like any other file content. The scanning engine can use signature-based detection, heuristic analysis, and behavioral analysis to detect any malicious content within the ADS.

4. **Monitoring and Real-Time Detection**: By continuously monitoring file operations in real-time, the hypervisor-based anti-virus can detect and respond to the creation of malicious ADS. This real-time monitoring ensures that any attempt to hide malware in ADS is immediately flagged and addressed.

Implementation Considerations

1. **Integration with NTFS APIs**: The hypervisor must integrate with NTFS APIs to access and enumerate ADS. This involves using low-level file system calls that can retrieve information about all data streams attached to a file.

2. **Performance Optimization**: Scanning ADS can introduce additional overhead. To mitigate this, the anti-virus solution can implement selective scanning, focusing on files and processes that exhibit suspicious behavior. Caching previously scanned files and their ADS can also reduce redundant scans.
3. **Security and Isolation**: The hypervisor provides an isolated environment for scanning, preventing malware from tampering with the anti-virus processes. This isolation enhances security by ensuring that even if the operating system is compromised, the hypervisor-based anti-virus remains effective.
4. **User and Kernel Mode Coordination**: Effective communication between user-mode components (such as the anti-virus engine and GUI) and kernel-mode components (such as the hypervisor) is essential. This coordination ensures that ADS scanning is integrated seamlessly into the overall anti-virus operations.

A hypervisor-based anti-virus solution can effectively scan Alternate Data Streams (ADS) by leveraging its low-level access and control over system operations. By intercepting file operations, accessing ADS information through NTFS APIs, and scanning ADS for malicious content, the hypervisor-based anti-virus can detect and mitigate threats hidden in ADS. This approach provides a robust and comprehensive defense against sophisticated malware techniques that exploit ADS to evade detection.

7

Related Works

7.1 Traditional AV and EDR

Traditional Antivirus (AV) software and modern Endpoint Detection and Response (EDR) solutions play crucial roles in the cyber security landscape. Both are designed to identify and mitigate malicious programs, but they operate in slightly different manners and scopes. This section delves into how these solutions work, particularly focusing on their use of API hooking to monitor and identify suspicious behavior in running applications.

7.1.1 Traditional Antivirus (AV) Solutions

Traditional AV software is designed to detect and remove malware from computer systems. These programs typically rely on a combination of signature-based detection and heuristic analysis.

1. **Signature-Based Detection**: This method involves scanning files and comparing their signatures (unique sequences of bytes) against a database of known malware signatures. When a match is found, the AV software flags the file as malicious and takes appropriate action, such as quarantining or deleting the file. While effective, this method relies on regular updates to the signature database to recognize new threats.

2. **Heuristic Analysis**: To complement signature-based detection, AV software employs heuristic analysis to identify new or modified malware. This involves examining the behavior and structure of files to detect suspicious activities that are indicative of malware, such as unusual file manipulations, code obfuscation, or unauthorized network communications. Heuristic analysis helps AV software detect previously unknown threats, often referred to as zero-day threats.

7.1.2 Endpoint Detection and Response (EDR) Solutions

EDR solutions extend beyond traditional AV by offering more comprehensive monitoring and response capabilities. EDR focuses on real-time detection, investigation, and remediation of threats across an organization's endpoints.

1. **Real-Time Monitoring**: EDR solutions continuously monitor endpoints for suspicious activities. This involves collecting and analyzing data from various sources, including system logs, network traffic, and application behavior. By maintaining constant vigilance, EDR solutions can detect and respond to threats more quickly than traditional AV software.

2. **Behavioral Analysis**: EDR solutions employ advanced behavioral analysis techniques to identify anomalies that may indicate malicious activity. This involves establishing a baseline of normal behavior for each endpoint and then flagging deviations from this baseline. For instance, if an application suddenly starts accessing sensitive files or communicating with unknown external servers, the EDR solution will alert administrators to a potential threat.

3. **Incident Response**: One of the key advantages of EDR is its ability to facilitate incident response. When a threat is detected, EDR solutions provide detailed forensic data to help security teams understand the scope and impact of the incident. They also offer tools to

contain and remediate the threat, such as isolating compromised endpoints, terminating malicious processes, and rolling back changes made by malware.

7.1.3 API Hooking in AV and EDR

Both AV and EDR solutions commonly use API hooking as a technique to monitor running applications for suspicious behavior. API hooking involves intercepting calls to system functions (APIs) made by applications and redirecting them to custom monitoring code. This allows security software to observe and analyze the behavior of applications in real-time.

1. **Implementation of API Hooking**: To implement API hooking, security software injects a small piece of code into the target application or the operating system's API functions. This code intercepts API calls and redirects them to the security software, which can then inspect the parameters and behavior of the call before allowing it to proceed. This technique is particularly useful for monitoring key activities such as file access, network communication, and process creation.

2. **Benefits of API Hooking**: API hooking provides several advantages for security monitoring. By observing API calls, security software can detect suspicious activities that may indicate malware, such as attempts to modify system files, access sensitive data, or communicate with command-and-control servers. Additionally, API hooking allows for more granular monitoring compared to traditional methods, as it can capture detailed information about the behavior of individual applications.

3. **Challenges of API Hooking**: Despite its effectiveness, API hooking also presents some challenges. Hooking API calls can introduce performance overhead, as each intercepted call must be processed by the security software. Additionally, sophisticated malware can employ techniques to detect and evade API hooks, making it more difficult for security software to monitor their behavior. To address these challenges, modern AV and EDR solutions often use a combination of API hooking, kernel-level monitoring, and other techniques to achieve comprehensive threat detection.

Traditional AV software and modern EDR solutions play critical roles in the cyber security landscape, each with its strengths and limitations. While traditional AV relies on signature-based detection and heuristic analysis, EDR extends these capabilities with real-time monitoring, behavioral analysis, and incident response. API hooking is a common technique used by both AV and EDR solutions to monitor running applications for suspicious behavior, providing a valuable tool for detecting and mitigating threats. Despite its challenges, API hooking remains an essential component of effective security monitoring, helping to protect systems and data from a wide range of malicious activities.

7.2 Limitations of API Hooking

7.2.1 Concealment by Sophisticated Malware

One of the main challenges of API hooking is its vulnerability to advanced evasion techniques used by sophisticated malware. Malware developers are well aware of the methods employed by security solutions and often design their malicious programs to bypass detection. For instance, some malware directly invokes system calls instead of using high-level API functions. This approach allows the malware to avoid detection since API hooks are typically placed at the API level rather than at the system call level.

Furthermore, certain malware can actively scan its own code and memory to detect the presence of hooks set by security software. If it finds such hooks, the malware can alter its behavior to evade detection or even disable the hooks altogether. Code obfuscation is another technique frequently used by malware to make its analysis more challenging. By encrypting or packing the code, malware can remain hidden from security tools, making it difficult to identify and hook specific API calls. Additionally, many forms of malware employ anti-debugging techniques to detect if they are being monitored or analyzed. Upon detecting the presence of a debugger or a security tool, the malware may modify its behavior to avoid detection.

7.2.2 Inability to Intercept Deeper System Calls

API hooking primarily operates at the user-mode level, intercepting calls made by applications to high-level API functions. However, it struggles to intercept system calls that occur at deeper levels within the operating system. Many critical operations take place at the kernel level, where API hooks are less effective. To monitor these operations, security solutions must implement kernel-mode drivers, which are more complex and pose greater stability and security risks. Kernel-mode drivers have access to low-level system resources and can intercept system calls at the kernel level, but developing and maintaining them introduces potential issues.

Another challenge with API hooking is the performance overhead it can introduce. Intercepting and analyzing API calls can slow down the system, particularly if hooks are placed on frequently used API functions. This performance overhead can affect the user experience, and developers of security software must find a balance between thorough monitoring and acceptable system performance. Advanced malware, such as rootkits and hypervisor-based malware, can operate at levels below the operating system, making them extremely difficult to detect with traditional API hooking techniques. These types of malware can intercept and manipulate system calls before they reach the operating system, effectively bypassing user-mode hooks.

7.2.3 Limited Visibility into Encrypted or Packed Code

API hooking also faces challenges when dealing with encrypted or packed code. Malware often uses encryption or packing to conceal its true nature and behavior, remaining in an unreadable state until it is unpacked or decrypted at runtime. This obfuscation makes it difficult for API hooks to analyze the malicious code before it executes. Since the true behavior of the malware is only revealed at runtime, static analysis techniques are insufficient. Dynamic analysis, which involves monitoring the code as it runs, becomes necessary. However, dynamic analysis is more complex and resource-intensive.

To effectively analyze encrypted or packed code, security software must implement techniques to unpack or decrypt the code at runtime. This requires sophisticated methods to handle the various packing and encryption schemes used by malware authors.

While API hooking is a valuable technique for monitoring and intercepting suspicious behavior in running applications, it has several limitations. Sophisticated malware can employ techniques such as direct system call invocation, inline hook detection, code obfuscation, and anti-debugging methods to evade detection. Additionally, API hooking struggles to intercept deeper system calls at the kernel level, faces performance overhead challenges, and has limited visibility into encrypted or packed code. To address these limitations, security solutions must combine API hooking with other advanced techniques, such as kernel-mode drivers, behavior-based analysis, and dynamic unpacking methods, to provide comprehensive threat detection and response capabilities.

7.3 Hypervisor-Based AV Architectures

Hypervisor-based antivirus (AV) solutions represent an advanced approach to malware detection and system security. Leveraging the capabilities of hypervisors, these solutions can facilitate kernel-level system call hooking, offering enhanced visibility and control over the operating system's activities. This section delves into the architecture design, isolation mechanisms, and performance considerations of hypervisor-based AV solutions.

7.3.1 Architecture Design

Hypervisor-based AV solutions rely on a virtualized environment where a hypervisor, also known as a virtual machine monitor (VMM), manages one or more virtual machines (VMs). The hypervisor operates at a higher privilege level than the guest operating systems, providing a strategic vantage point for monitoring and controlling system activities.

1. **Hypervisor Layer**: The hypervisor is the core component of the architecture. It sits between the hardware and the guest operating systems, managing the execution of multiple VMs. This layer allows the AV solution to intercept and monitor system calls at the kernel level, providing deep visibility into the activities of the guest OS.

2. **Guest Operating System**: The guest OS runs within a VM managed by the hypervisor. It functions as a typical operating system, running applications and handling user interactions. The AV solution operates alongside the guest OS but has the advantage of being monitored and controlled by the hypervisor.

3. **Monitoring and Control Modules**: These modules are embedded within the hypervisor to track system calls and other critical activities. They can inspect the behavior of applications and processes running within the guest OS, allowing for real-time detection and intervention of suspicious activities.

4. **Communication Interface**: The interface facilitates communication between the hypervisor and the AV management console. This console provides a user-friendly interface for administrators to monitor system status, configure security policies, and respond to detected threats.

7.3.2 Isolation Mechanisms

One of the significant advantages of hypervisor-based AV solutions is their ability to provide strong isolation between the security functions and the guest OS. This isolation enhances security and stability by preventing malware from tampering with the AV solution.

1. **Separation of Privileges:** The hypervisor operates at a higher privilege level than the guest OS. This separation ensures that even if malware gains root access within the guest OS, it cannot affect the hypervisor or its monitoring modules.
2. **Protected Memory Spaces:** Hypervisor-based AV solutions can allocate dedicated memory spaces for their operations. These memory areas are inaccessible to the guest OS, ensuring that critical AV functions and data remain secure from malware attacks.
3. **Integrity Monitoring:** The hypervisor can continuously monitor the integrity of the guest OS and its components. By maintaining a baseline of trusted states, the AV

solution can detect unauthorized changes and take corrective actions to maintain system integrity.

7.3.3 Performance Considerations

While hypervisor-based AV solutions offer significant security advantages, they also introduce performance considerations that must be managed to ensure a balance between security and system performance.

1. **Overhead Management**: Running AV functions within the hypervisor layer introduces some overhead, as every system call and critical operation is monitored. Efficient design and optimization of the hypervisor and monitoring modules are crucial to minimize performance impact.
2. **Resource Allocation:** The hypervisor must efficiently allocate resources such as CPU, memory, and I/O to ensure that the AV functions do not adversely affect the performance of the guest OS. Advanced resource management techniques, including dynamic allocation and prioritization, can help maintain optimal performance.
3. **Latency Reduction:** Monitoring and intercepting system calls can introduce latency in the execution of applications and processes. Hypervisor-based AV solutions must implement strategies to minimize latency, such as selective monitoring and prioritization of critical system calls.

Hypervisor-based AV architectures provide robust security by facilitating kernel-level system call hooking and offering enhanced visibility into the activities of the guest OS. Their architecture design leverages the hypervisor's privileged position to monitor and control system operations effectively. Isolation mechanisms ensure that the AV solution remains secure and tamper-proof, even in the presence of sophisticated malware. However, performance considerations must be carefully managed to balance security with system efficiency. Through efficient design, resource management, and latency reduction strategies, hypervisor-based AV solutions can provide comprehensive security without compromising system performance.

7.4 Dynamic Process Monitoring Engines

Dynamic process monitoring engines play a crucial role in modern cyber security strategies, particularly in real-time threat detection and response. Unlike traditional static analysis methods, which examine code in a dormant state, dynamic process monitoring involves observing and analyzing processes as they execute. This approach enables security systems to detect and respond to malicious activities in real-time, offering significant advantages in combating sophisticated and fast-evolving threats.

7.4.1 Development of Dynamic Process Monitoring Engines

Developing dynamic process monitoring engines involves integrating several key components and techniques aimed at providing comprehensive real-time analysis of system activities. At the heart of these engines are behavioral analysis frameworks, which track the behavior of applications and processes during execution. These frameworks establish baselines of normal behavior and flag deviations that may indicate malicious activity. For instance, unexpected network connections, abnormal file modifications, and unusual system calls are scrutinized closely.

To facilitate this monitoring, the system must be instrumented to capture relevant events. This is typically achieved through hooks at various levels, such as API hooks, system call hooks, and inline hooks. These hooks intercept important function calls and system interactions, allowing the engine to gather detailed information about the behavior of processes. Real-time data collection and analysis are critical components of dynamic process monitoring. The engine continuously collects data from the running system, including memory access patterns, CPU usage, disk I/O, and network activity. Advanced data analysis techniques, including machine learning algorithms and statistical models, are then applied to identify patterns indicative of malicious behavior. This real-time analysis ensures immediate detection and response to threats.

Effective dynamic process monitoring engines are also integrated with other components of the security infrastructure, such as intrusion detection systems (IDS), antivirus software, and endpoint detection and response (EDR) platforms. This integration facilitates the sharing of threat intelligence and coordinated response efforts across the security ecosystem.

7.4.2 Implementation of Dynamic Process Monitoring Engines

Implementing dynamic process monitoring engines requires careful consideration of several factors to ensure both efficacy and minimal performance impact. One of the main challenges is optimizing performance. Monitoring processes dynamically can introduce performance overhead, as the system must handle additional data collection and analysis tasks. Developers mitigate this by selectively focusing on high-risk processes and critical system activities, thereby reducing overall resource consumption.

Another critical challenge is minimizing false positives—legitimate activities mistakenly identified as threats. Sophisticated filtering techniques and continuous refinement of behavioral baselines help improve the accuracy of threat detection, ensuring that genuine threats are identified without inundating administrators with false alarms. Scalability and flexibility are also essential. Dynamic process monitoring engines must be scalable to handle environments of varying sizes, from individual devices to large enterprise networks. The engine should be flexible enough to adapt to different operating systems, application types, and network configurations. This adaptability ensures broad applicability and effectiveness across diverse IT landscapes.

Ensuring that monitoring activities do not interfere with legitimate user and process activities is critical. The engine must operate transparently, maintaining user privacy and process integrity while still capturing necessary data. Isolation techniques, such as sandboxing, are often employed to segregate monitoring activities from normal system operations.

7.4.3 Significance in Real-Time Threat Detection and Response

Dynamic process monitoring engines are vital for real-time threat detection and response strategies. They provide immediate threat identification by monitoring processes as they execute, allowing the system to detect and respond to threats instantaneously. This real-time capability is essential for mitigating the damage caused by fast-spreading malware, ransomware, and other types of attacks that can rapidly compromise systems.

These engines are adept at detecting advanced threats that evade traditional static analysis techniques. For example, polymorphic malware, which changes its code to avoid signature-based detection, can be identified based on its behavior rather than its static characteristics. Dynamic monitoring engines provide comprehensive visibility into system activities, capturing a wide range of events and interactions that static methods might miss. This holistic view enables security teams to understand the full scope of an attack, from initial entry to lateral movement and data exfiltration.

Dynamic process monitoring enables proactive defense mechanisms. When a threat is detected, the system can take immediate actions such as isolating the affected process, terminating malicious activities, and alerting security personnel. This proactive approach helps prevent further spread and escalation of attacks.

Dynamic process monitoring engines are pivotal in modern cyber security, offering real-time threat detection and response capabilities essential for defending against sophisticated and rapidly evolving threats. Their development involves integrating behavioral analysis frameworks, instrumentation techniques, and real-time data collection and analysis, all optimized for performance and accuracy. Implementation considerations include minimizing performance impact, reducing false positives, ensuring scalability, and maintaining process isolation. The significance of these engines lies in their ability to provide immediate threat identification, detect advanced threats, offer comprehensive visibility, and enable proactive defense strategies.

7.5 Performance Evaluations

Evaluating the performance of security mechanisms is essential to understanding their impact on system resources and their effectiveness in detecting threats. This section reviews existing studies comparing user-mode API hooking with kernel-level system call hooking enabled by hypervisors, focusing on resource utilization, processing overhead, and effectiveness in malware detection.

7.5.1 Comparing User-Mode API Hooking and Kernel-Level System Call Hooking

User-mode API hooking and kernel-level system call hooking are two different approaches to monitoring and intercepting system activities. User-mode API hooking operates at the application level, intercepting API calls made by programs. In contrast, kernel-level system call hooking, often enabled by hypervisors, operates at a deeper level within the operating system, intercepting system calls made by all processes.

7.5.2 Resource Utilization

One of the primary factors in performance evaluations is resource utilization. User-mode API hooking generally consumes fewer system resources compared to kernel-level hooking. This is because user-mode hooks operate within the context of individual applications, requiring less overhead for each interception. However, the trade-off is that user-mode hooks can miss lower-level activities that occur beneath the API layer.

Kernel-level system call hooking, enabled by hypervisors, offers a more comprehensive view of system activities but at the cost of higher resource utilization. This method monitors all system calls, providing a more detailed analysis of system behavior. However, because it operates at the kernel level, it requires more CPU and memory resources to manage the additional processing workload. Hypervisors add another layer of abstraction, further increasing resource demands.

7.5.3 Processing Overhead

Processing overhead is another critical factor. Studies show that user-mode API hooking introduces less processing overhead compared to kernel-level system call hooking. User-mode hooks can quickly intercept and analyze API calls without significantly delaying the execution of applications. This makes user-mode hooking more suitable for systems where performance is a critical concern.

On the other hand, kernel-level system call hooking involves more complex operations. Intercepting system calls at the kernel level requires the hypervisor to handle additional context switches and memory management tasks, which can introduce noticeable delays in system performance. While this overhead is justified by the increased visibility and security it provides, it can be a limiting factor for performance-sensitive environments.

7.5.4 Effectiveness in Malware Detection

Effectiveness in malware detection is perhaps the most crucial aspect of these performance evaluations. User-mode API hooking is effective in detecting a wide range of threats, especially those that rely on high-level API functions. However, sophisticated malware can evade detection by bypassing API calls and directly invoking system calls or employing techniques to disable or evade user-mode hooks.

Kernel-level system call hooking, facilitated by hypervisors, offers superior effectiveness in detecting such advanced threats. By intercepting system calls at a lower level, this method can catch malicious activities that bypass user-mode hooks. Hypervisors can monitor all system interactions, providing a more comprehensive security solution. This makes kernel-level hooking particularly effective against rootkits, stealth malware, and other sophisticated threats that operate at or near the kernel level.

Performance evaluations of user-mode API hooking and kernel-level system call hooking reveal a trade-off between resource utilization, processing overhead, and effectiveness in malware detection. User-mode API hooking is more efficient in terms of resource usage and processing speed, making it suitable for performance-critical environments. However, it may be less effective against sophisticated malware that operates at a lower level. Kernel-level system call hooking, enabled by hypervisors, offers enhanced security by providing deeper visibility into system activities and better detection of advanced threats. However, this comes at the cost of higher resource utilization and increased processing overhead. The choice between these methods depends on the specific security requirements and performance constraints of the environment.

8

Conclusion

This project explored a new way to improve computer security using hypervisor-based antivirus (AV) solutions. Traditional methods, like user-mode API hooking, are not always effective against advanced cyber threats. These older methods often miss malware that operates deep within the system, leaving computers vulnerable.

Hypervisor-based AV solutions, which work at a deeper level in the system (the kernel level), offer a stronger defense. By using hypervisors, which control virtual machines and operate below the operating system, these AV solutions can monitor and control system activities more effectively. This deeper monitoring allows them to catch malicious activities that traditional methods might miss.

We looked at how these hypervisor-based AV solutions work, including their design and how they access hidden parts of files, like Alternate Data Streams (ADS). We also considered the impact on system performance and found ways to optimize these solutions to reduce any slowdown.

Through real-world examples and tests, we showed that hypervisor-based AV solutions detect and stop threats better than traditional methods. By monitoring system calls at the kernel level, they provide real-time protection and a stronger defense against malware.

In summary, this project demonstrated that hypervisor-based AV solutions are a powerful tool for improving computer security. They overcome the limitations of traditional AV methods by providing deeper and more effective monitoring. This new approach offers better protection for both individuals and organizations against today's complex cyber threats.

9

QR Code

10

References

[1] Osaghae OE, Egbokhare, FA and Chiemeke SC. June 2020. “DESIGN OF GENERIC ANTIVIRUS SYSTEM”. IEEE

[2] Vittorio Orbinato, Marco Carlo Feliciano, Domenico Cotroneo, Roberto Natella. Nov 2023. “Laccolith: Hypervisor-Based Adversary Emulation with Anti-Detection” IEEE

[3] Dayantha Jayarathna, Udaya Tupakula, Vijay Varadharajan. January 2015. “Hypervisor-based security architecture to protect web applications”.

[4] Daniel Quist, Lorie Liebrock, Joshua Neil. April 2010. “Improving antivirus accuracy with hypervisor assisted analysis”.

[5] Yevgeniy Miretskiy, Abhijith Das, Charles P. Wright, and Erez Zadok. 2004. “Avfs: An On-Access Anti-Virus File System”. Stony Brook University.

[6] https://www.geeksforgeeks.org/protection-ring/

[7] https://jamcyber.com/blog/cyber-insights/next-generation-antivirus-vs-traditional-antivirus/

[8] Empirical Study on Anti-Virus Architecture for Container platforms

[9] Intel® Virtualization Technology for Connectivity

[10] https://www.ibm.com/think/insights/virtualization-benefits

[11] https://xopero.com/blog/en/intro-to-the-virtualization-types-benefits-and-disadvantages-you-should-consider/

[12] "Malware Analyst's Cookbook and DVD: Tools and Techniques for Fighting Malicious Code" by Michael Ligh, Steven Adair, Blake Hartstein, and Matthew Richard.

[13] "Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software" by Michael Sikorski and Andrew Honig.

[14] "Rootkits: Subverting the Windows Kernel" by Greg Hoglund and James Butler

[15] "The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory" by Michael Hale Ligh, Andrew Case, Jamie Levy, and AAron Walters.

[16] "Virtualization and Forensics: A Digital Forensic Investigator's Guide to Virtual Environments" by Diane Barrett and Gregory Kipper