

天津大学

**Breaking Theoretical Limits: The Gap Between
Virtual Network Interface Cards and Physical
Network Cards**

学 校: 天津大学

学 院: 网络安全学院

姓 名: 谢远峰

背景介绍

虚拟化技术提供了创建管理虚拟化资源（如虚拟机、虚拟网络）的技术，提供像 Open vSwitch¹（SDN）工具一样实现相邻虚拟机之间的通信。

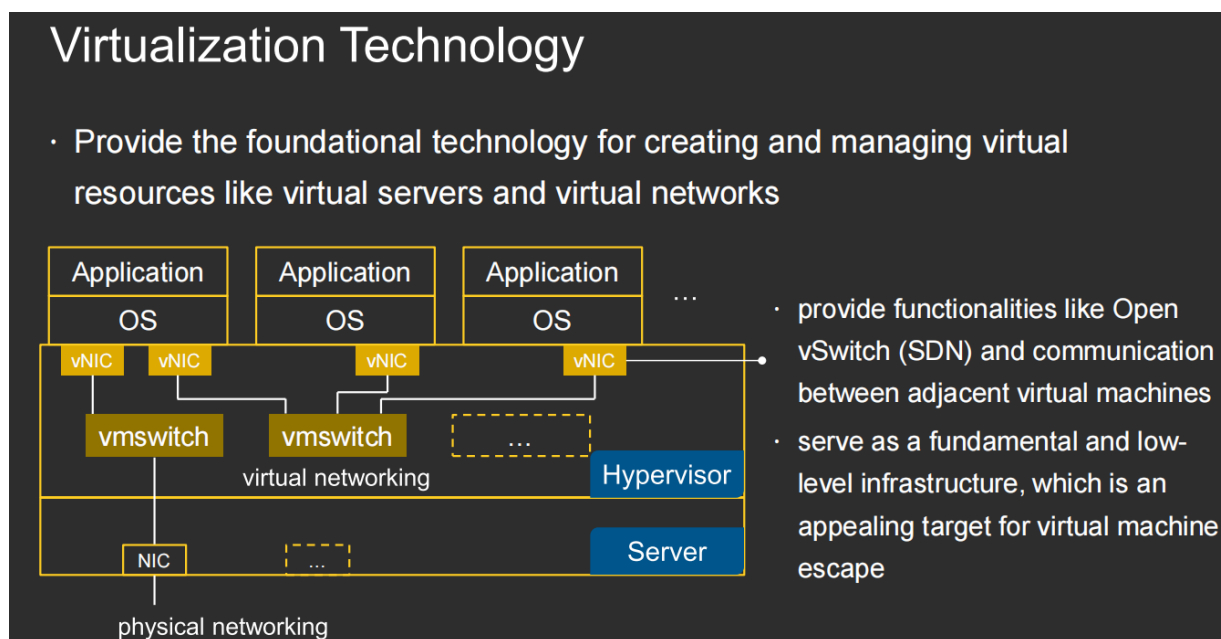


Figure 1: 虚拟化技术

虚拟网卡（vNIC）作为一个基本的低级基础设施，是虚拟机逃逸的一个有效的目标。Hypervisor 可以为每个虚拟机创建一个或多个 vNIC，站在虚拟机的角度，这些 vNIC 等同于物理的网卡。vNIC 经过 vmSwitch 通过物理 Server 的物理网卡访问外部的物理网络。虚拟网卡（例如 E1000, E1000e, VMXNET, VMXNET2, VMXNET3）在虚拟环境中利用软件（如 VMware、VirtualBox、KVM 等）模拟物理网卡特性。

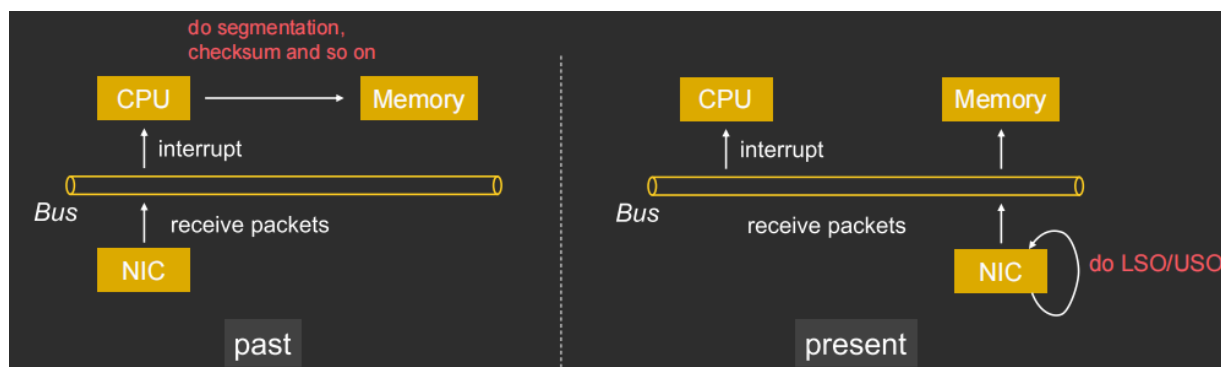


Figure 2: 网卡作用

在计算机系统中，CPU 通过中断请求通知操作系统需要进行 I/O 操作。操作系统将 CPU 请求的数据映射到内存中的特定区域，这个区域通常称为 I/O 内存映射（I/O memory mapping）。然后，这些数据通过计算机系统的总线传输到相应的外部设备，或者从外部设备接收数据并传输到 CPU 指定的内存地址。总线在这个过程中扮演了连接 CPU、内存和外部设备的通信路径，实现了数据的传输和通信。

¹ Open vSwitch 是一个用 C 语言开发的多层虚拟交换机，主要目标是实现一个支持标准管理接口的，开放转发功能以支持编程扩展和控制的交换机平台，适合在虚拟机环境中用作虚拟交换机。

传统虚拟网卡接受数据包后，需要中断 CPU 进程，由 CPU 执行数据包的分段和校验和计算等任务。当前虚拟网卡可以承担部分 CPU 负载，先行执行数据包的分段、校验和计算任务，能够有效处理小型数据包，针对大型数据包，虚拟网卡会将任务挂载到硬件网卡执行。**(UDP Segmentation Offload, USO)**对 UDP 包发送，超过 MTU(最大传输单元) 大小的报文不需要在协议栈分段，直接挂载到网卡，由网卡硬件实现分段，降低 CPU 负载。**(Large Send Offload, LSO)**是一种在高带宽网络中用于减少 CPU 使用率和增加发送吞吐量的技术，该技术通过网卡对过大的 TCP 数据进行分段。

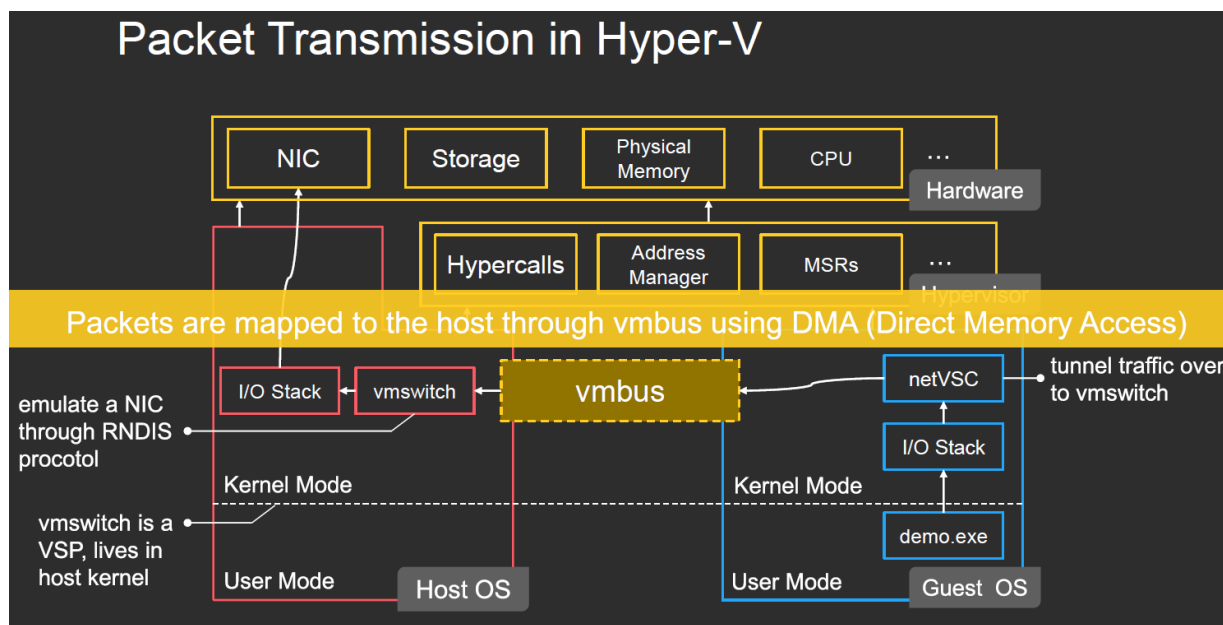


Figure 3: Hyper-V 虚拟机架构数据包传输示意图

如上图所示为 Hyper-V 架构下数据包传输的完整过程。在 Hyper-V 架构下，客户机在在用户层发送的数据包经过内核态的 I/O 网络堆栈。**netVSC¹**驱动程序利用 RNDIS 协议模拟出的虚拟网卡接受 I/O 网络堆栈中的数据包，通过 **VMBus²**总线隧道传输到根分区的 **VSP³⁴**中的 **vmswitch⁵**程序，**vmSwitch** 将数据包通过根主机的 I/O 网络堆栈传送到物理网卡，并通过物理网卡发送到外部网络。

1 NetVSC 是一个独立的驱动程序，类似于 virtio 和 vmxnet3。使用 NetVSC 需要将相关的 VMBUS 设备绑定到 Hyper-V 的用户空间 I/O 设备驱动程序 (uio_hv_generic)。默认情况下，所有 netvsc 设备都将绑定到 Linux 内核驱动程序。

2 VMBus 是一种基于通道的通信机制，用于在具有多个活动虚拟化分区的系统上进行分区间通信和设备枚举。此过程使用 DMA（直接内存访问）技术。

3 VSP（虚拟服务提供程序）。VSP 调用对应的设备驱动，直接访问硬件，中间不需要 Hypervisor 的帮助。

4 VSP 与 VSC:父分区托管虚拟服务提供程序 (VSP)，虚拟服务提供程序通过 VMBus 进行通信以处理子分区的设备访问请求。子分区托管虚拟服务使用方 (VSC)，虚拟服务使用方通过 VMBus 将设备请求重定向到父分区中的 VSP

5 vmswitch(虚拟交换机):包括以编程方式管理且可扩展的功能,可将虚拟机同时连接到虚拟网络和物理网络。此外, Hyper-V 虚拟交换机提供了安全、隔离和服务级别的策略执行。

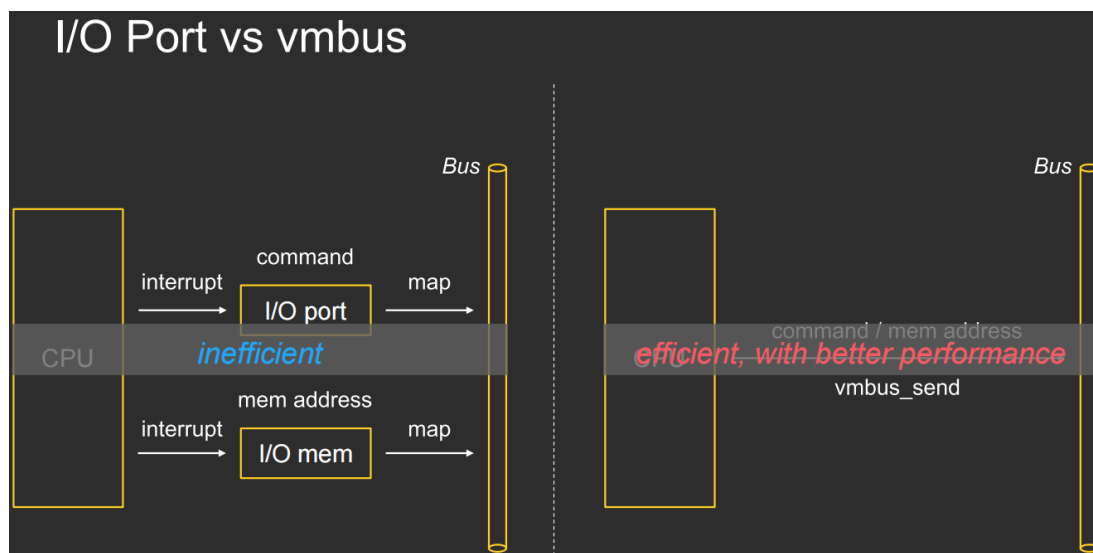


Figure 4: I/O 端口与 VMBus 通信对比图

如上图所示，vmbus 相比 I/O Port 在性能和效率上有明显的优势。这是因为 vmbus 使用了更高效的通信机制，通过使用内置的 `vmbus_send` 函数¹，减少了 CPU 中断和内存映射，从而提高了系统性能。而 I/O Port 则需要更多的 CPU 中断和内存映射，这可能会降低系统性能。

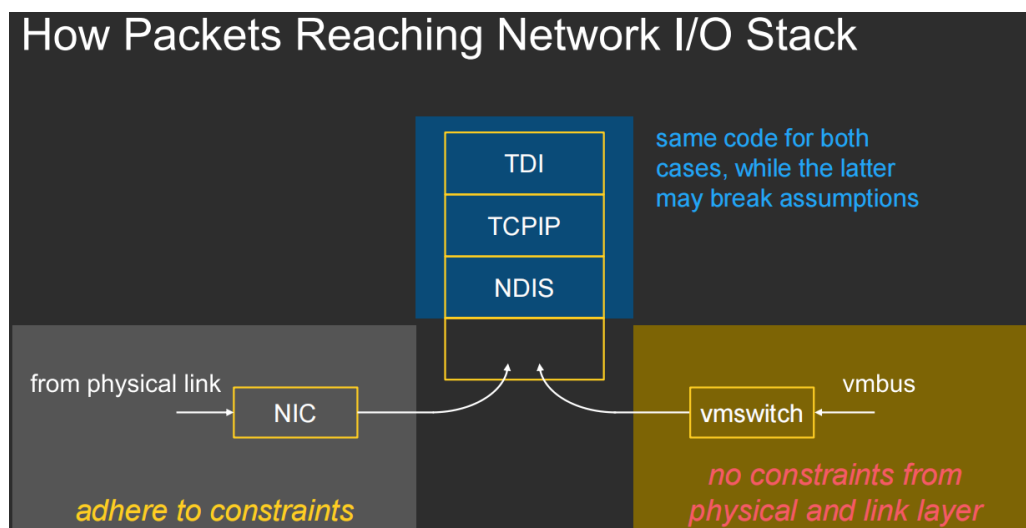


Figure 5: 数据包到达网络 I/O 堆栈方式示意图

如上图所示，从物理链路获取的数据包首先通过物理链路到达网络接口卡（NIC）。在这个阶段，数据包需要遵守一定的约束条件。在虚拟环境中，数据包通过虚拟机交换机 `vmswitch` 和 `vmbus` 进行处理。这个过程没有来自物理和链路层的约束，因此可能会打破某些假设。

数据包进入网络 I/O 堆栈，该堆栈由三个层级组成：NDIS、TCPIP 和 TDI。NDIS（**Network Driver Interface Specification**）是一个应用程序接口（API），用于网络设备驱动程序。NDIS 定义了一组标准的接口，网络驱动程序可以使用这些接口与上层的网络传输协议进行通信。TCPIP（**Transmission Control Protocol/Internet Protocol**）是一组用于互联网上数据交换的协议。TCP 负责在数据传送时分割成合适的块，然后由 IP 负责将这些块发送到正确的目的

¹ 基本定义：vmbus_send 函数用于将数据发送到数据包缓冲区或外部数据内存描述符列表 (MDL)。功能：vmbus_send 函数将数据与 VMBus 数据包对象相关联，该对象在事务的整个生存期内表示数据包。使用场景：当需要在 VSP（虚拟服务提供程序）和 VSC（虚拟服务使用方）之间传递消息时，可以使用 vmbus_send 函数。这种情况通常发生在虚拟机和 Hyper-V 主机之间的通信过程中 1。

地。**TDI (Transport Driver Interface)** 是一个抽象层，位于网络协议栈的传输层和网络设备驱动程序之间。TDI 提供了一种机制，使得多个传输协议可以共享同一个网络接口。

在虚拟环境中，通过 VMBus 传输到 vmswitch 的数据包需要经过以下阶段：

1. 将消息转换成数据包
2. 进入协议处理程序，并注册到 NDIS 中
3. 通过一系列的过滤，验证，转发等操作，到达 VmsVmNicPvtPacketForward() 函数¹
4. 调用协议栈上的相应处理程序发送数据包

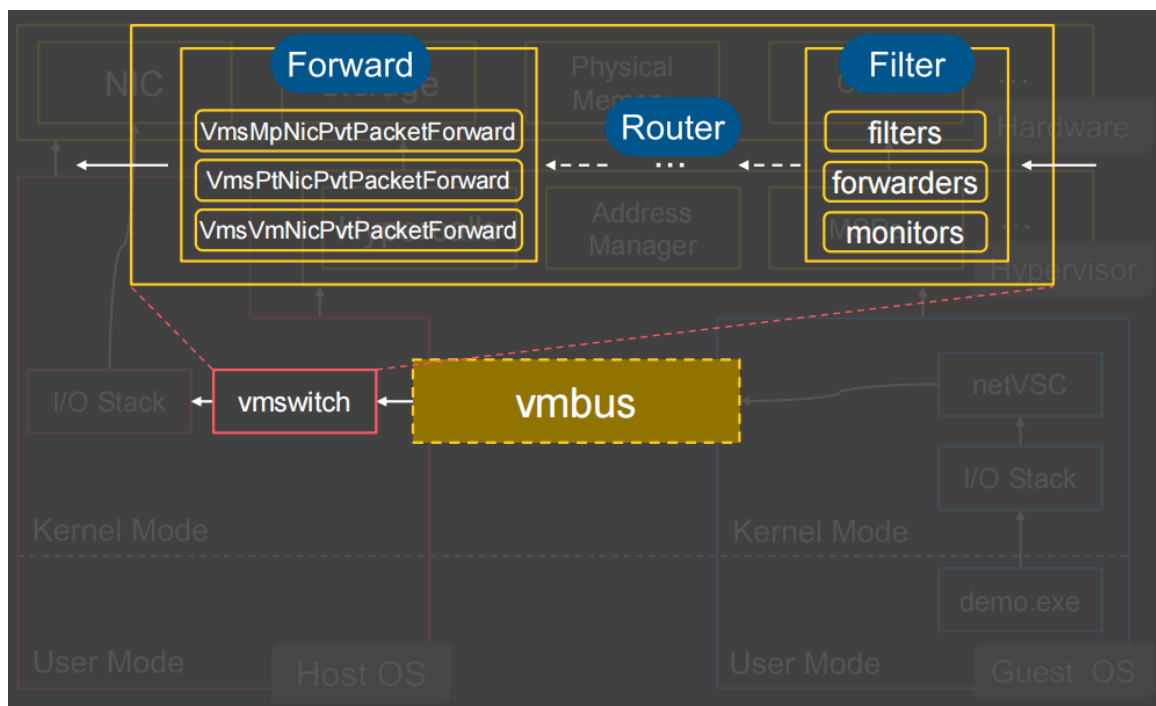


Figure 6: VmSwitch 数据包处理过程示意图²³

如上图所示，vmswitch 获取的数据包依次需要经过 Filter，Router，Forwarder 三个阶段。Filter 阶段用于数据包的过滤，包括数据包的验证，转发，恶意监控等；Router 阶段用于数据包的路由，包括数据包的转发，路由表的更新等；Forwarder 阶段用于数据包的发送，包括虚拟网卡发送对象，多处理器网卡发送对象，直通模式网卡发送对象。

¹ VmsVmNicPvtPacketForward: VmSwitch Virtual Network Interface Card Private Packet Forwarding

² VmsMpNicPvtPacketForward: VmSwitch Multi-Processor Network Interface Card Private Packet Forwarding

³ VmsPtNicPvtPacketForward: VmSwitch Passthrough Network Interface Card Private Packet Forwarding

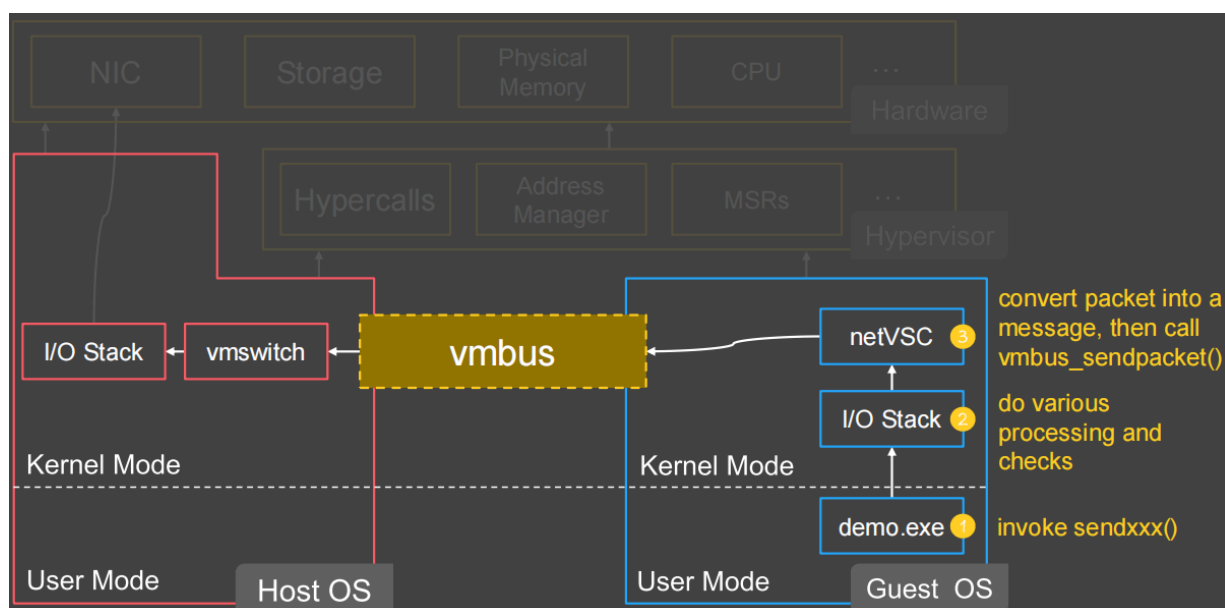


Figure 7: 正常数据包处理流程示意图

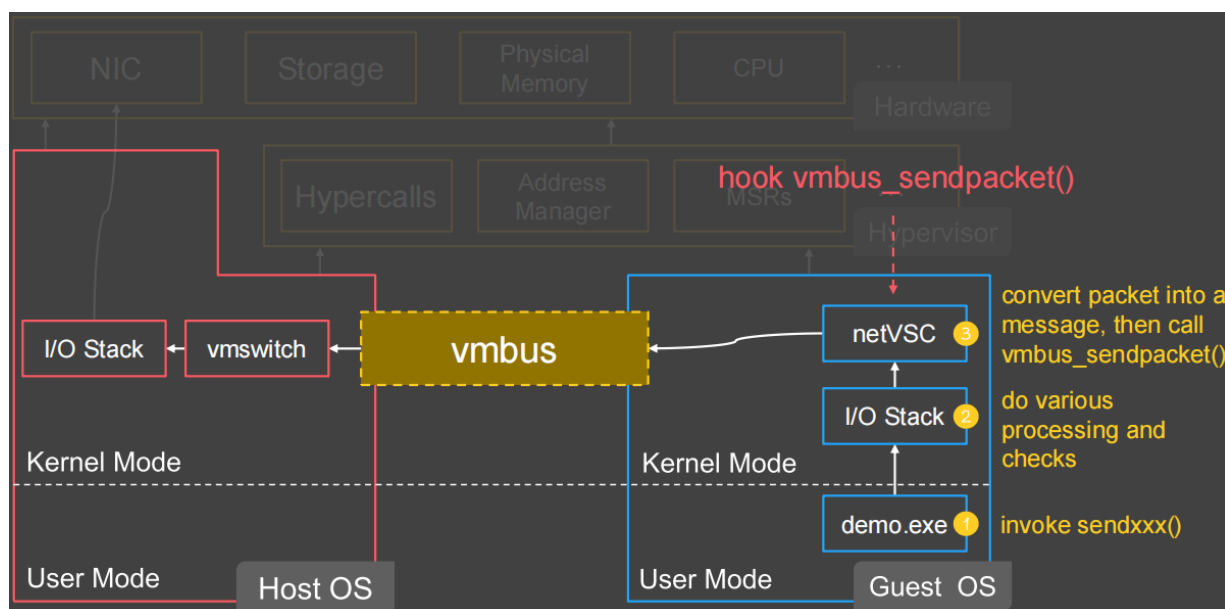


Figure 8: 异常数据包处理流程示意图

如图7所示，用户在客户虚拟机上发送消息，首先经过用户态下程序的 `sendxxx` 函数，进入内核态中的 I/O 堆栈进行消息的处理和校验检查，而后进入 `netVSC` 驱动程序，将消息分割转换为数据包，通过 `vmbus_sendpacket` 函数进入 `vmbus` 通信线。

如图8所示，为了实现异常消息的发送工具，且绕过内核态 I/O 堆栈的检查，利用 Hook¹ 可以劫持 `vmbus_sendpacket` 函数，将错误数据包进行发送。

¹ 钩子(Hook)，是 Windows 消息处理机制的一个平台，应用程序可以在上面设置子进程以监视指定窗口的某种消息，而且所监视的窗口可以是其他进程所创建的。当消息到达后，在目标窗口处理函数之前处理它。钩子机制允许应用程序截获处理 window 消息或特定事件。钩子实际上是一个处理消息的程序段，通过系统调用，把它挂入系统。

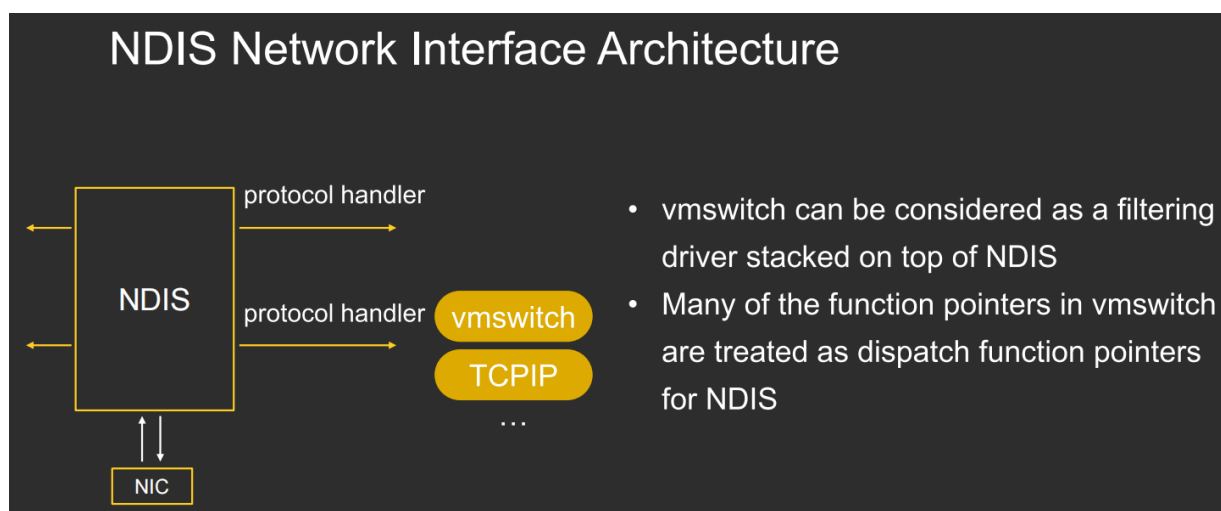


Figure 9: NDIS (Network Driver Interface Specification) 网络接口架构

如图 9 所示，NDIS 作为应用程序接口，对接上层 TCPIP 与下层 NIC 或 vmswitch。vmswitch 可以被看作为基于 NDIS 的过滤驱动程序，在 vmswitch 中，许多函数指针可被认为是 NDIS 的分派函数指针，这些函数指针可以被劫持，从而实现对数据包的劫持。

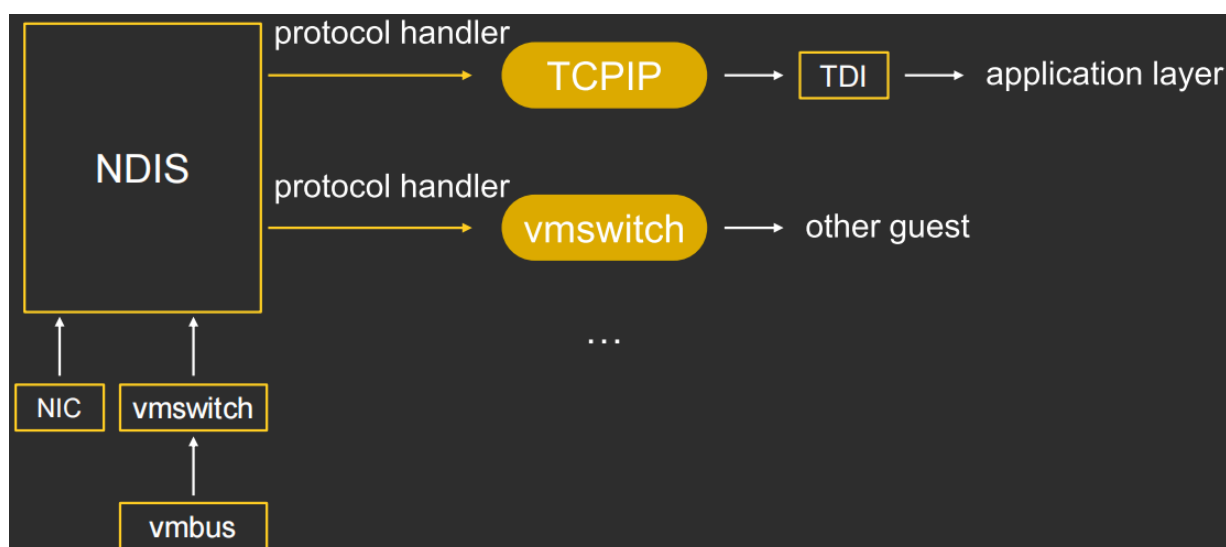


Figure 10: 数据包经过 NDIS 接口的处理流程

如图 10 所示，下层分别接受物理链路层中物理网络卡中的数据包和虚拟环境中 vmswitch 中的数据包，协议处理器可以通过 vmswitch 调用 NDIS 接口转发数据包到其他客户机，也可以通过 TCP/IP 协议传递到 TDI，最终到达应用层。

同样的实现方式适用于不同来源的传入数据包，而实现过程中隐藏的先决条件可能会被破坏。

实验流程

1. 测试目标准备

1. 选取测试目标：Windows 平台负责网络通信的驱动（例. `tcpip.sys`, `vmswitch.sys`）
2. 测试目标逆向：利用 IDApro 针对测试目标进行逆向，（可能进行相关代码优化）获取逆向函数并导出 C 代码

2. LLM（大语言模型）处理，多 Agent 处理

1. Agent1，函数名解读：针对逆向获取的函数名，扩充获取函数名的全称，再针对全称进行可能涉及的网络协议映射，进而实现 RFC 文档编号的对应（例. `CTlsExt::ParseTlsExtensions->CTlsExtensionsParser::ParseTransportLayerSecurityExtensions->TLS 协议->RFC8446`）
2. Agent2，代码阅读：输入每个函数，利用 Agent1 解读的函数名称和相关协议信息，解读代码，根据 RFC 文档对代码进行语义抽象（例. `Ipv6pHandleRouterAdvertisement` 函数（涉及 RFC8200<IPv6>, RFC2461<邻居发现>）->依次进行参数检查，IPV6 类型校验，数据包长度获取，缓冲区建立，消息选项解析->邻居发现与更新）
3. Agent3，RFC 文档章节匹配：针对 Agent2 得出的代码切片进行 RFC 文档章节定位（例. `Ipv6pHandleRouterAdvertisement` 函数消息选项解析切片-> RFC8200 第 4 节（IPv6），RFC2461 第 4.2 节（RA 消息））
4. Agent4，逆向代码 RFC 文档校验：针对 Agent3 给出的切片匹配 RFC 信息进行 RFC 文档校验，给出可疑的不一致行为（例. RFC8200 第 4 节（IPv6）提到建议每个 IPv6 选项最多出现一次，如果选项出现多次必须能够正确解析）

3. 模糊测试平台

1. Fuzz 平台：Jackalope（黑盒二进制模糊测试工具）原因：支持黑盒二进制，支持自动化分析和报告，支持生成高度结构化的输入。
2. 利用 Agent4 得出的不一致点书写报文构造规则（例. 构建一个能够包含多个 IPv6 选项的报文看看能不能正常解析，用 Jackalope 报告查看结果）

Motivation

虚拟化技术是云计算的关键组成部分，为创建和管理虚拟资源（如虚拟服务器和虚拟网络）提供了基础技术。在云环境中，虚拟机通常配备虚拟网卡（**Network Interface Cards, NICs**），使虚拟机能够连接到网络并与其他计算机或设备通过互联网进行通信。考虑到虚拟网卡作为基础且低级的基础设施，它已成为安全研究人员感兴趣的目标。

在使用 **Hyper-V** 中的虚拟网卡时，其实现应当与物理网络卡的实现保持一致。因此，虚拟网卡实现中固有的隐藏前提条件也应当保持不变。具体来说，物理网络卡的 IP 数据包存在长度限制。单个 IP 数据包（未合并的分段数据包）的长度不能小于 28 字节，也不能大于 65535 字节。然而，由于虚拟网卡与物理网络卡之间的差异，这些长度限制可以被突破。也就是说，协议栈实现中的某些隐藏前提条件可能不成立，如果被滥用，可能导致服务器漏洞。

攻击目标关注于 **Hyper-V** 客户环境中虚拟网卡模块。了解 **Windows Hyper-V** 虚拟机网络通信架构和 IP 数据包传输机制，全面比较虚拟网卡和真实网络卡之间的 IP 数据包处理例程。通过利用 IP 数据包处理例程中的漏洞作为案例研究，为针对虚拟网卡进行代码审查或模糊测试提供新的方向。

背景知识

链路层协议头<Ethernet II 以太帧>（源 MAC 地址，目标 MAC 地址），网络层协议标识
网络层协议头<IPv6>，长度，下一个扩展头标识，跳数限制，源 IP 地址，目标 IP 地址
逐跳选项扩展报头
其它非上层扩展头选项（Option），[目的选项扩展报头,路由扩展报头,分片扩展报头,授权扩展报头,封装安全有效载荷扩展报头]
上层扩展报头（如 TCP/UDP/ICMP 等）

IPv6 报文简易构造。一个 IPv6 报文包含几个部分：链路层协议头，网络层协议头，其他网络层扩展头，上层（应用层）扩展报头

漏洞说明

Target	ID	Bounty	Type	Status
Windows TCP/IP	CVE-2021-24074	\$100,000	Integer overflow	Fixed
Windows Hyper-V	CVE-2022-30223	\$25,000	Out Of Bounds	Fixed
Windows Hyper-V	CVE-2023-36908	\$25,000	Out Of Bounds	Fixed
Windows VMWare	MSRC Case 71449-1	0	Null pointer deference	Submitted
Windows VMWare	MSRC Case 71449-2	0	Null pointer deference	Submitted


```
The IPv6 header [IPv6] has a 16-bit Payload Length field and,
therefore, supports payloads up to 65,535 octets long. This document
specifies an IPv6 hop-by-hop option, called the Jumbo Payload option,
that carries a 32-bit length field in order to allow transmission of
IPv6 packets with payloads between 65,536 and 4,294,967,295 octets in
length. Packets with such long payloads are referred to as
"jumbograms".
```

Figure 13: RFC2675 (IPv6 Jumbograms) 中提到 Ipv6 报文可以构造超过 65536 的长度

CVE-2021-24074

链接	https://msrc.microsoft.com/update-guide/en-US/advisory/CVE-2021-24074
攻击目标 ¹	Network-TCP/IP: 漏洞组件与网络堆栈绑定，潜在的攻击者范围超出了其他列出的选项，甚至包括整个互联网。这种漏洞通常被称为“远程可利用”，可以理解为攻击在协议级别上是可利用的，跨越了一个或多个网络跳数（例如，跨越一个或多个路由器）。
攻击复杂性 ²	低:特定的访问条件或特殊情况并不存在。攻击者可以预期在漏洞组件上获得可重复的成功。
赏金	100,000\$
CVE 编号	2021-24074
报告时间	2021-02-09
影响平台	Windows Server、Windows 10、Windows8.1、Windows 7
影响版本	未知

表 3: CVE-2021-24074 漏洞信息

CVE-2021-24074 漏洞复现搭建步骤：

1. windows 11 系统开启虚拟化 Intel VT-x/EPT³，关闭基于虚拟化的安全性⁴
2. 虚拟机 VMware16.1.2 安装与激活⁵；windows server 2019 虚拟机安装并关闭更新，开启共享文件夹与物理机联通；拷贝 centos7 镜像以及程序源文件到共享文件夹⁶
3. windows server 2019 虚拟机安装利用 IE 安装 Edge，hyper-v 服务⁷
4. windows server 2019 虚拟机安装 wireshark，IDA8.4，WinDbg⁸
5. 在 hyper-v 服务管理器中安装 centos7 虚拟机安装并配置；网络联通测试；用 wireshark 抓包⁹
6. windows server 2019 虚拟机安装 WinScp 服务实现文件共享；触发漏洞¹⁰

漏洞说明：

Windows TCP/IP 驱动 tcpip.sys 在解析过长(超过 65536 bytes) ICMPv6 路由器通告 (Router Advertisement) 报文时存在整数溢出漏洞，攻击者可通过构造特制的 ICMPv6 路由器通告报文，触发漏洞导致蓝屏崩溃。

攻击者利用通过在路由器通告的 IPV6 头中插入 Jumbo Option(一个可以将 Ipv6 报文长度扩展超过 65536 的选项)，使路由器通告报文超过 Ipv6 报文长度限制，导致 tcpip.sys 解析时发生整数溢出，从而导致蓝屏崩溃。

1 该指标反映了利用漏洞的可能性。基础分值越高，攻击者利用漏洞组件的距离就越远（逻辑上和物理上）
 2 该指标描述了攻击者在利用漏洞时必须存在的超出其控制范围的条件。这些条件可能需要收集有关目标的更多信息或计算异常。对于利用漏洞而言，评估此度量不包括对用户交互的任何要求。如果攻击成功需要特定的配置，则应该根据漏洞组件处于该配置下来评分基本度量。
 3 win11 家庭版 使用 vm 进行虚拟化的设置（此平台不支持虚拟化的 Intel VT-x/EPT。）
 4 关闭 Windows10/11“基于虚拟化的安全性”的四种方法,采用最后一种
 5 阿里云盘 VMware 程序下载；阿里云盘 VMware 程序激活码
 6 Windows 10 2004 19041.867；windows server 关闭更新；vmware 开启共享文件夹；Centos 7.7.1908 x64
 7 信任站点添加允许 Edge 安装包下载；Winserver2019；
 8 拿到 tcpip.sys 文件进行逆向；C:/Windows/MiniDump/*.dmp 文件进行 WinDbg 解析
 9 最小配置+调试工具+兼容性库+开发工具；Linux 安装教程；安装 ifconfig
 10 hyper 文件传输；WinScp 使用教程；关闭 centos 安全启动，my_hid.c 修改 source_mac/ip,des_mac/ip, my_hid.c 及相关文件拷贝到虚拟机 centos 中，make/insmod my_hid.ko（一定时间后触发 Windows 虚拟机蓝屏崩溃）

CVE-2021-204074 相关资料

- [讲解视频](#)
- [IPV6 Router Advertisement](#)

```

__int64 __fastcall Ipv6pHandleRouterAdvertisement(/*..args..*/) {
    // <1> NetBuffer
    struct _NET_BUFFER *v9;
    // <2> Option Total Length (max 65535)
    unsigned __int16 v21;
    // <3> Option pointer
    KIRQL *v28;
    // <4> Option Length
    unsigned __int16 v29;
    // ... Validate the Router Advertisement ...
    while (1) {
        // <4-1> Get current Option pointer and current Option Length
        v28 = (KIRQL *)NdisGetDataBuffer(v9, 2u, &v231, 1u, 0);
        v29 = 8 * v28[1];
        // ...
        // Move forward to the next option. (1)
        // Keep track of the Parsed Length, so we can use it below to back up.
        NdisAdvanceNetBufferDataStart(v9, v29, 0, 0);
        // ...
        // (2) <2-1> Option Total Length Update WITH OVERFLOW RISK
        v21 += v29; /*.....v21 VARIABLE OVERFLOW*/
    }
    // ...
    NdisRetreatNetBufferDataStart(v9, v21, 0, NetioAllocateMdl);
    // ...
}

```

代码 1: 逆向漏洞代码展示。变量 v21 记录了 Option 的总长度，在第 5 行被初始化为上限为 65536 的短整型变量。在 while 循环结构中，变量 v21 在第 21 行被记录当前 Option 长度的变量 v29 不断递增。由于没有进行检查，因此存在整数溢出的风险。

代码信息补充

```

// microsoft/win32metadata/generation/WinSDK/RecompiledIdlHeaders/shared/netiodef.h
typedef struct _ICMP_HEADER {
    UINT8 Type;           // Type of message (high bit zero for error messages).
    UINT8 Code;           // Type-specific differentiator.
    UINT16 Checksum;      // Calculated over ICMP message and IPvx pseudo-header.
} ICMP_HEADER, *PICMP_HEADER;
typedef struct _ICMP_MESSAGE {
    ICMP_HEADER Header;
    union {
        UINT32 Data32[1];    // Type-specific field.
        UINT16 Data16[2];    // Type-specific field.
        UINT8 Data8[4];      // Type-specific field.
    } Data;
} ICMP_MESSAGE, *PICMP_MESSAGE;
typedef ICMP_MESSAGE ICMPV6_MESSAGE, *PICMPV6_MESSAGE;
typedef struct nd_router_advert {
    ICMPV6_MESSAGE nd_ra_hdr;
    UINT32 nd_ra_reachable;    // Reachable Time.
    UINT32 nd_ra_retransmit;   // Retransmit Timer.
    // Could be followed by options.
} ND_ROUTER_ADVERT_HEADER, *PND_ROUTER_ADVERT_HEADER;
// ND_OPTION_TYPE: Define values for neighbor discovery option type.
typedef enum {
    ND_OPT_SOURCE_LINKADDR = 1,
    ND_OPT_MTU = 5,
    ND_OPT_PREFIX_INFORMATION = 3,
    ND_OPT_ROUTE_INFO = 24,
    // ...
} ND_OPTION_TYPE, *PND_OPTION_TYPE;

```

代码 2: sizeof(ND_ROUTER_ADVERT_HEADER) AND Switch Opcode

```

BOOLEAN Ipv6pParseTlvOption(IN PNET_BUFFER NetBuffer, OUT PCHAR Type, OUT PUSHORT Length){
    ND_OPTION_HDR Buffer, *Option;
    if (NetBuffer->DataLength < sizeof(ND_OPTION_HDR)) {
        // Insufficient data buffer for a valid TLV option.
        return FALSE;
    }
    Option = NetioGetDataBuffer(NetBuffer, sizeof(ND_OPTION_HDR), &Buffer, 1, 0);
    *Length = ((USHORT) Option->nd_opt_len) * 8;
    if ((*Length == 0) || (*Length > NetBuffer->DataLength)) {
        // Invalid option length.
        return FALSE;
    }
    *Type = Option->nd_opt_type;
    return TRUE;
}

typedef struct nd_opt_hdr {
    UINT8 nd_opt_type;
    UINT8 nd_opt_len;           // In units of 8 octets.
    // Followed by option specific data.
} ND_OPTION_HDR, *PND_OPTION_HDR;

```

代码 4: Ipv6pParseTlvOption

```

typedef struct _NET_BUFFER {
    union {
        struct {
            NET_BUFFER *Next;
            MDL *CurrentMdl;
            ULONG CurrentMdlOffset;
            union {
                ULONG DataLength;
                SIZE_T stDataLength;
            };
            MDL *MdlChain;
            ULONG DataOffset;
        };
        SLIST_HEADER Link;
        NET_BUFFER_HEADER NetBufferHeader;
    };
    USHORT ChecksumBias;
    USHORT Reserved;
    NDIS_HANDLE NdisPoolHandle;
    PVOID NdisReserved[2];
    PVOID ProtocolReserved[6];
    PVOID MiniportReserved[4];
    PHYSICAL_ADDRESS DataPhysicalAddress;
    union {
        NET_BUFFER_SHARED_MEMORY *SharedMemoryInfo;
        SCATTER_GATHER_LIST *ScatterGatherList;
    };
} NET_BUFFER, *PNET_BUFFER;

typedef struct _NET_BUFFER NET_BUFFER, *PNET_BUFFER;
typedef struct _NET_BUFFER_LIST_CONTEXT NET_BUFFER_LIST_CONTEXT, *PNET_BUFFER_LIST_CONTEXT;
typedef struct _NET_BUFFER_LIST NET_BUFFER_LIST, *PNET_BUFFER_LIST;

typedef uint32_t NDIS_HANDLE, *PNDIS_HANDLE;

```

代码 4: NET_BUFFER

CVE-2022-30223

链接	https://msrc.microsoft.com/update-guide/en-US/advisory/CVE-2022-30223
攻击目标 ¹	Network-TCP/IP :漏洞组件与网络堆栈绑定，但攻击在协议层面上被限制在逻辑上相邻的拓扑中。这意味着攻击必须从同一个共享的物理网络（例如，蓝牙或 IEEE 802.11）或逻辑网络（例如，本地 IP 子网）中发起，或者从一个安全的或其他有限的管理域内（例如，MPLS，安全 VPN 到管理网络区域）发起。
攻击复杂性 ²	低:特定的访问条件或特殊情况并不存在。攻击者可以预期在漏洞组件上获得可重复的成功。
CVE 赏金	25,000\$
报告时间	2022-07-22
影响平台	Windows Server、Windows 10、Windows8.1、Windows 7
影响版本	未知

表 4: CVE-2021-24074 漏洞信息

CVE-2022-30223 漏洞复现搭建步骤：

1. windows 11 系统开启虚拟化 Intel VT-x/EPT³，关闭基于虚拟化的安全性⁴
2. 虚拟机 VMware16.1.2 安装与激活⁵；windows server 2019 虚拟机安装并关闭更新，开启共享文件夹与物理机联通；拷贝 centos7 镜像以及程序源文件到共享文件夹⁶
3. windows server 2019 虚拟机安装利用 IE 安装 Edge，hyper-v 服务⁷
4. windows server 2019 虚拟机安装 wireshark，IDA8.4，WinDbg⁸
5. 在 hyper-v 服务管理器中安装 centos7 虚拟机安装并配置；网络联通测试；用 wireshark 抓包⁹
6. windows server 2019 虚拟机安装 WinScp 服务实现文件共享；上传文件¹⁰
7. 禁用网路接口 UDP,TCP 校验和；编译文件；触发漏洞

漏洞说明：

Hyper-V 客户机发送 DNS UDP 报文时，缓存内没有目标地址缓存，Windows 主机进行 ARP 报文应答时，需要首先对 UDP 报文进行校验和计算，而校验和的计算需要确定 UDP 报文长度以及 UDP 报文头在报文中的偏移位置。正确的校验逻辑应该是将 UDP 报文头偏移位置与实际的数据长度进行比较，这样可以确保 UDP 报头偏移不会超过数据包的实际长度，然而当前实现是将 UDP 报文头偏移位置与 Windows 格式（RNDIS）报文长度进行比较，而 Windows 格式（RNDIS）报文长度可以被攻击者控制，从而绕过正确的校验。

- 1 该指标反映了利用漏洞的可能性。基础分值越高，攻击者利用漏洞组件的距离就越远（逻辑上和物理上）
- 2 该指标描述了攻击者在利用漏洞时必须存在的超出其控制范围的条件。这些条件可能需要收集有关目标的更多信息或计算异常。对于利用漏洞而言，评估此度量不包括对用户交互的任何要求。如果攻击成功需要特定的配置，则应该根据漏洞组件处于该配置下来评分基本度量。
- 3 win11 家庭版使用 vm 进行虚拟化的设置（此平台不支持虚拟化的 Intel VT-x/EPT。）
- 4 关闭 Windows10/11“基于虚拟化的安全性”的四种方法,采用最后一种
- 5 阿里云盘 VMware 程序下载；阿里云盘 VMware 程序激活码
- 6 Windows 10 2004 19041.867；windows server 关闭更新；vmware 开启共享文件夹；Centos 7.7.1908 x64
- 7 信任站点添加允许 Edge 安装包下载；Winserver2019；
- 8 拿到 tcpip.sys 文件进行逆向；C:/Windows/MiniDump/*.dmp 文件进行 WinDbg 解析
- 9 最小配置+调试工具+兼容性库+开发工具；Linux 安装教程；安装 ifconfig
- 10 hyper 文件传输；WinScp 使用教程；关闭 centos 安全启动，my_hid.c 修改 source_mac/ip,des_mac/ip, my_hid.c 及相关文件拷贝到虚拟机 centos 中，make/insmod my_hid.ko（一定时间后触发 Windows 虚拟机蓝屏崩溃）

CVE-2022-30223 逆向代码

```
__int64 __fastcall VmsNblHelperCreateCloneNbl(/*..args..*/){
    // <1> Bytes to copy
    int v24, v60;
    // <2> Maybe DataAdjust flag
    char v26;
    // ...
    // <2-1> CheckSum bit get
    v23 = SrcNetBufferList->NetBufferListInfo[0];
    v26 = v23 != 0i64 ? a6 : 0; // flag?
    if ( v26 ) {
        // TcpCheckSum bit check
        if ( ( (unsigned __int8)v23 & 4 ) != 0 ) {
            // ...
        LABEL_80:
            // <1-2> Bytes to copy Update
            v60 = v24; /*.....(2)CONSTANT ASSIGNMENT 34*/
            NdisAdvanceNetBufferListDataStart(SrcNetBufferList, (unsigned __int16)v24, 0, 0i64);
            goto LABEL_82;
        }
        // <1-1> UdpCheckSum bit check, Pass and Jump to LABEL_80, Bytes to copy Update
        if ( ( (unsigned __int8)v23 & 8 ) == 0 ) {
            v24 = 34; /*.....(1)CONSTANT ASSIGNMENT 34*/
            goto LABEL_80;
        }
        // ...
    }
    // ...
    LABEL_82:
    // ...
    if ( v26 ) {
        // ...
        NetBufferListContext = NdisRetreatNetBufferListDataStart( \
            CloneNetBufferList, (unsigned __int16)v24, 0, 0i64, 0i64);
        if ( NetBufferListContext >= 0 ) {
            // ...
            while ( 1 ) {
                /* (3) The function copy NetBuffer from one to another
                The Third Arg v60 is BytesToCopy constant With INFO LEAK RISK*/
                NetBufferListContext = NdisCopyFromNetBufferToNetBuffer(\
                    v48, 0, (unsigned __int16)v60, Alignment, 0, &BytesCopied);
                // ...
            }
        }
    }
}
```

代码 5: 逆向代码 VmsNblHelperCreateCloneNbl 裁剪

```
_IRQL_requires_max_(DISPATCH_LEVEL)
NDIS_EXPORTED_ROUTINE
NDIS_STATUS
NdisCopyFromNetBufferToNetBuffer(
    _In_ NET_BUFFER                                *Destination,
    _In_ ULONG                                     DestinationOffset,
    _In_ ULONG                                     BytesToCopy,
    _In_ NET_BUFFER const                         *Source,
    _In_ ULONG                                     SourceOffset,
    _Out_ ULONG                                    *BytesCopied
);
```

代码 6: NdisCopyFromNetBufferToNetBuffer 函数声明

上述代码展示的是 VmsNblHelperCreateCloneNbl 漏洞触发的过程，NdisCopyFromNetBufferToNetBuffer 函数将一段长缓存进行拷贝。

```

typedef struct _NET_BUFFER {
    union {
        struct {
            NET_BUFFER *Next;
            MDL *CurrentMdl;
            ULONG CurrentMdlOffset;
            union {
                ULONG DataLength;
                SIZE_T stDataLength;
            };
            MDL *MdlChain;
            ULONG DataOffset;
        };
        SLIST_HEADER Link;
        NET_BUFFER_HEADER NetBufferHeader;
    };
    USHORT ChecksumBias;
    USHORT Reserved;
    NDIS_HANDLE NdisPoolHandle;
    PVOID NdisReserved[2];
    PVOID ProtocolReserved[6];
    PVOID MiniportReserved[4];
    PHYSICAL_ADDRESS DataPhysicalAddress;
    union {
        NET_BUFFER_SHARED_MEMORY *SharedMemoryInfo;
        SCATTER_GATHER_LIST *ScatterGatherList;
    };
} NET_BUFFER, *PNET_BUFFER;

typedef struct _NET_BUFFER NET_BUFFER, *PNET_BUFFER;
typedef struct _NET_BUFFER_LIST_CONTEXT NET_BUFFER_LIST_CONTEXT, *PNET_BUFFER_LIST_CONTEXT;
typedef struct _NET_BUFFER_LIST NET_BUFFER_LIST, *PNET_BUFFER_LIST;

typedef uint32_t NDIS_HANDLE, *PNDIS_HANDLE;

```

代码 7: NET_BUFFER 结构体声明及宏定义声明

```

typedef enum _NDIS_NET_BUFFER_LIST_INFO {
    TcpIpChecksumNetBufferListInfo = 0, // IMPORTANT
    TcpLargeSendNetBufferListInfo = 1,
    IpSecOffloadV1NetBufferListInfo = 2,
    IpSecOffloadV2NetBufferListInfo = 3,
    TcpReceiveNoPushNetBufferListInfo = 4,
    TcpReceiveCoalescedInfo = 5,
    ClassificationHandleNetBufferListInfo = 6,
    NblOriginalInterfaceIfIndexNetBufferListInfo = 7,
    NblReAuthWfpFlowContextNetBufferListInfo = 8,
    MaxNetBufferListInfo = 9
} NDIS_NET_BUFFER_LIST_INFO, *PNDIS_NET_BUFFER_LIST_INFO;

typedef union _NDIS_TCP_IP_CHECKSUM_NET_BUFFER_LIST_INFO {
    struct {
        ULONG IsIPv4 : 1;
        ULONG IsIPv6 : 1;
        ULONG TcpChecksum : 1; // IMPORTANT
        ULONG UdpChecksum : 1; // IMPORTANT
        ULONG IpHeaderChecksum : 1;
        ULONG Reserved : 27;
    } Transmit;
    struct {
        ULONG TcpChecksumFailed : 1;
        ULONG UdpChecksumFailed : 1;
        ULONG IpChecksumFailed : 1;
        ULONG Reserved : 29;
    } Receive;
    ULONG Value; // 0x90 字节
} NDIS_TCP_IP_CHECKSUM_NET_BUFFER_LIST_INFO, *PNDIS_TCP_IP_CHECKSUM_NET_BUFFER_LIST_INFO;

```

代码 8: NET_BUFFER_LIST 声明

```

__int64 __fastcall VmsVmNicPvtConvertRndisPacketToNbl(/*...args...*/)
{
    // The function begins by setting up some local variables and checking if a1 is non-zero
    // to determine which log to use (VmsIfrLog or VmsIfrNicLog). It checks if the size of
    // the packet *((_DWORD *)a2 + 5) is less than 14 bytes. If it is, the function logs
    // an error and returns with an error code -1073741811.
    // ...
    v3 = VmsIfrLog;
    if ( a1 ){v3 = VmsIfrNicLog;}
    // ...
    v5 = *((_DWORD *)a2 + 5) < 0xEu;
    // Check if packet size is less than the minimum required (14 bytes)
    if ( v5 ){
        // LOG ERROR
    }
    // ...
    // Allocates a NetBufferList using NdisAllocateNetBufferList. Checks if the
    // allocation was successful. If not, logs an error and returns with an error code -1073741670.
    NetBufferList = NdisAllocateNetBufferList(v12, 0x158u, 0);
    // ...
    // if fail, log error
    // if success, start to convert
    if ( NetBufferList )
    {
        // ...
        // Initializes the context associated with the NetBufferList, setting various fields and pointers.
        v19 = (__int64)&v16->Context->ContextData[v16->Context->Offset];
        // ...
        // Sets up the packet data and headers, including retreating the data start by the header size
        // and retrieving the data buffer.
        NdisRetreatNetBufferDataStart(FirstNetBuffer, v14, 0, 0i64);
        DataBuffer = NdisGetDataBuffer(FirstNetBuffer, v14, 0i64, 1u, 0);
        // ...
        // Iterates through the MDLs (Memory Descriptor Lists) to copy data from the MDLs
        // to the data buffer in the NetBufferList. Handles mapping of pages if necessary
        // and checks various conditions to ensure correct data copying.
        // ...
        if ( v39 )
        {
            NetBufferList = v16;
            while ( 1 )
            {
                // ...
                v45 = *((_BYTE *)v43 + 10) & 5) != 0
                    ? (char *)v43[3]
                    : (char *)MmMapLockedPagesSpecifyCache((PMDL)v43, 0, MmCached, 0i64, 0, 0x40000000u);
                if ( !v45 )
                    break;
                // ...
            }
            // ...
        }
        // ...
        // Finalization: Once the data is copied, the function sets various fields in
        // the NetBufferList based on the packet descriptor. Checks for conditions to
        // ensure data integrity and logs errors if necessary.
    }
    // ...
    // ERROR code
    // can not allocate netbufferlist
    OffsetAfterIpHeader = -1073741670;
    // ...
LABEL_176:
    // return error
    // ...
    // return success
    return (unsigned int)OffsetAfterIpHeader;
}

```

代码 9: VmsVmNicPvtConvertRndisPacketToNbl 函数裁剪

```

__int64 ValidatePerPacketInfoUnion(/*...args...*/){
    // arg: a1 validate option
    // arg: unsigned int *a3, ---transmit.tcp_header_offset
    // ...
    // csum_info is passed in one of the parameters, let's say a5.
    // unsigned int v20; // r10d
    // CSUM_INFO *csum_info = (CSUM_INFO *)a5;

    switch (a1) {
    case 0:
        if ((unsigned __int64)(unsigned int)_mm_cvtsil28_si32(_mm_srli_si128(*a15, 8)) - a16 >= 4) {
            __int64 v19 = a16 + 4;
            if ((unsigned __int64)(a16 + 4) <= 0xFFFFFFFF) {
                unsigned int v20 = *(_DWORD *)((unsigned int)a16 + a15->m128i_i64[0]);
                // error
                if ((HIWORD(v20) & 0x3FF) + 20 > a2){ return 0x6000000000i64;}
                // csum_info->transmit.tcp_header_offset = v20; // Setting the value
                *a3 = v20;
            }
            return v19;
        }
        return 0x2000000000i64;
    // Other cases...
    default:
        return 0x3000000000i64;
    }
}

```

代码 10: ValidatePerPacketInfoUnion 函数裁剪

```

__int64 __fastcall RndisDevHostInternalParsePacketMessage(/*args.. */){
    // arg: unsigned int a2
    unsigned int ByteCount; // ebx
    struct _MDL *v7; // r15
    int v9; // edx
    unsigned int v10; // ebx
    char *MappedSystemVa; // rcx
    char *v12; // rax
    unsigned int v13; // r13d
    // ...
    v7 = (struct _MDL *)a2;
    // ...
    // original message information length
    v12 = (char *)v7 + (_QWORD)MappedSystemVa;
    v13 = *(_DWORD *)((char *)&v7->Next + (_QWORD)MappedSystemVa + 4);
    if ( v13 < 8 )
    {
        // error process
    }
    // ...
}

```

代码 11: RndisDevHostInternalParsePacketMessage

RndisDevHostInternalParsePacketMessage 函数处理原始报文信息，其中包括原始报文的长度。ValidatePerPacketInfoUnion 函数设置 TCP 报文头初始偏移长度。VmsVmNicPvtConvertRdisPacketToNbl 函数将客户机报文转换为 windows 格式下的 netbufflist 结构进行存储。

变量处理

```

use regex::Regex;
use std::fs::{File, OpenOptions};
use std::io::{BufRead, BufReader, Write};
use std::io;

fn process_file(input_path: &str, output_dir: &str, variables_output_dir: &str) -> io::Result<usize> {
    // 编译正则表达式
    let re = Regex::new(r"//-----\s*\(((\s*\w+)+)\)\s*--").unwrap();
    // 匹配变量声明的正则表达式
    let variable_declaration_re = Regex::new(r"; //").unwrap();

    // 创建输出目录 (如果不存在)
    // ...

    // 打开输入文件
    // ...

    // 逐行读取文件
    for line in reader.lines() {
        let line = line?;

        // 检查是否匹配正则表达式
        if re.is_match(&line) {
            // 关闭当前文件 (如果有)
            if let Some(mut file) = current_file.take() {file.flush()?;}
            // 创建新文件
            // ...
        }

        // 写入当前行到文件 (如果有打开的文件)
        if let Some(ref mut file) = current_file {
            writeln!(file, "{}", line)?;
        }
    }

    // 关闭最后一个文件
    if let Some(mut file) = current_file {
        file.flush()?;
    }

    // 处理生成的文件, 提取变量声明所在行
    for entry in std::fs::read_dir(output_dir)? {
        let entry = entry?;
        let path = entry.path();

        if path.is_file() {
            let file = File::open(&path)?;
            let reader = BufReader::new(file);
            // 文件命名
            // ...
            let mut variable_count = 0;

            // 读取文件
            // ...

            writeln!(variables_file, "total variable number: {}", variable_count)?;
            variables_file.flush()?;

            total_variable_count += variable_count;
        }
    }

    // 输出
    println!("处理文件 {} 完成, 生成 {} 个文件, 总变量个数为 {}", input_path, file_count, total_variable_count);
    Ok(total_variable_count)
}

```

代码 12: 利用正则表达式抽取函数名称构建函数切片, 统计函数切片数量; 针对函数切片进一步构造变量切片, 统计函数切片内变量的数量

```
PS C:\Users\xyf20\Desktop\test_rs\file_splitter> cargo run
Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.01s
Running `target\debug\file_splitter.exe`
处理文件 tcpip.txt 完成, 生成 5186 个文件, 总变量个数为 55612
处理文件 vmswitch.txt 完成, 生成 3452 个文件, 总变量个数为 42417
处理完成, 总变量个数为 98029
```


ONE-SHOT 提示词工程

以下是 **one-shot 提示词工程示例**，通过给定上下文要求，样例输入，样例输出得到准确的输出。

```
Prompt:
Classify the sentiment of each conversation in <<<CONVERSATIONS>>> as 'Positive' or 'Negative'. Give the sentiment
classifications without any other preamble text.
###
EXAMPLE CONVERSATIONS
[Agent]: Good morning, how can I assist you today?
[Customer]: This product is terrible, nothing like what was advertised!
[Customer]: I'm extremely disappointed and expect a full refund.

[Agent]: Good morning, how can I help you today?
[Customer]: Hi, I just wanted to say that I'm really impressed with your
product. It exceeded my expectations!

###
EXAMPLE OUTPUTS

Negative

Positive

###
<<<
[Agent]: Hello! Welcome to our support. How can I help you today?
[Customer]: Hi there! I just wanted to let you know I received my order, and
it's fantastic!
[Agent]: That's great to hear! We're thrilled you're happy with your purchase.
Is there anything else I can assist you with?
[Customer]: No, that's it. Just wanted to give some positive feedback. Thanks
for your excellent service!

[Agent]: Hello, thank you for reaching out. How can I assist you today?
[Customer]: I'm very disappointed with my recent purchase. It's not what I expected at all.
[Agent]: I'm sorry to hear that. Could you please provide more details so I can help?
[Customer]: The product is of poor quality and it arrived late. I'm really
unhappy with this experience.
>>>

ChatGpt 4o output result:
Positive

Negative
```

代码 13: 提示词工程示例：使用分隔符进行分段提示

CO-STAR 框架的使用

- 上下文 (Context): 提供任务背景信息，有助于 LLM 了解正在讨论的具体场景，确保响应相关
- 目标 (Objective): 定义 LLM 执行的任务，有助于 LLM 将其回应集中于实现该特定目标
- 风格 (Style): 指定写作风格，指导 LLM 以符合要求的方式和措辞做出回应。
- 语气 (Tone): 设定回应的态度，确保 LLM 的回应与预期的情绪或所需的情感背景产生共鸣。
- 受众 (Audience): 确定响应的目标受众，针对受众（例如某个领域的专家、初学者、儿童等）定制 LLM 的回应，可确保其在您所需的背景下是适当且易于理解的。
- 响应 (Response): 提供响应格式，确保 LLM 以下游任务所需的确切格式输出。示例包括列表、JSON、专业报告等。对于大多数以编程方式处理 LLM 响应以进行下游操作的 LLM 应用程序，JSON 输出格式将是理想的选择。

ONE-SHOT 漏洞定位与分析

```
#CONTEXT#
I am code security auditor and I have got the code of a C function.
#####

#OBJECTIVE#
I would like to perform a code audit (including data flow static analysis, control flow static analysis, etc.)
against function code <<<CODE>>> to see if there is any risk of being exploited by an attack (including variable
overflow, buffer overflow, reading illegal pointers, etc.). I want you to follow the following content step by
step to complete and do not output your internal execution process of the 1,2,3,4 step like "Parameters and
Return Values", "Declared Variables and Types", "Potential Vulnerability Issues", directly output the answer of
the step 5 after processing, starting with "Actual Vulnerability Issues", without other outputs, like "Summary",
"Recommendations".
1. Get the parameters and return values of a function, and infer the range of return values and parameters from
the types of function parameters and return values. If the variable types of function parameters and return
values are not C basic types, make an educated guess about the scope of the variable representation based on
your experience.
2. Obtain the variables declared in the body of the function and the types of the variables, and infer the scope
of the variables based on the types of the variables. If the type of the variable declared in the body of the
function is not one of the basic types of the C language, make an educated guess about the scope of the variable
based on your experience.
3. Trace the assignment of three types of variables: function parameters, function return values, and declared
variables within a function to determine potential vulnerabilities, including, but not limited to, variable
overflows, buffer overflows, and wild pointer reads.
4. Traverse the function code again and utilize the potential vulnerability information obtained in the third
step in conjunction with the vulnerability and the contextual content of the vulnerability in order to determine
if the vulnerability actually exists in the function, and delete the potential vulnerability information if it
does not meet the conditions triggered by obtaining the potential vulnerability information in the third step,
or else update the potential vulnerability information to the actual vulnerability information.
5. List actual vulnerability issues with details ranging from first to forth steps.
#####
#STYLE#
Academic Analysis Report
#####
# TONE #
Professional, technical
#####
#AUDIENCE#
Anyone who understands code security, vulnerability security.
#####
#RESPONSE EXAMPLE#
The "test" variable is an unsigned variable, data flow analysis reveals that the execution code "test +=
test2" ("test2" is also an unsigned variable), control flow analysis of the execution code context reveals that
the code above does not have a scope check on the "test" variable, and the "test" variable updated below the
execution code does not have an overflow check, so there is an actual vulnerability.
#####

# START ANALYSIS #
If you understand, ask me for my code.

#####
<<<
Ipv6pHandleRouterAdvertisement Function Trimed Code
>>>
```

代码 14: 针对裁剪后的 Ipv6pHandleRouterAdvertisement 函数模拟代码审计进行漏洞检测

- **CONTEXT:** 作为代码安全审计人员，拿到了一个 C 语言函数
- **OBJECTIVE:** 进行静态分析代码审计，查看是否有安全漏洞，按照步骤进行执行，省略前 4 步的执行内容，如不输出“Parameters and Return Values”“Declared Variables and Types”“Potential Vulnerability Issues”，输出第 5 步的结果，以“Actual Vulnerability Issues”开始，不输出“Summary”，“Recommendations”
 1. 获取函数参数及返回值二者的类型，结合 GPT 自身经验判断表示范围
 2. 获取函数体内声明变量的类型，结合 GPT 自身经验判断表示范围
 3. 跟踪函数参数、函数返回值和函数内声明变量这三种变量的赋值，以确定潜在漏洞，包括但不限于变量溢出、缓冲区溢出和野指针读取。
 4. 再次遍历函数代码，利用第三步获取的潜在漏洞信息，结合漏洞和漏洞的上下文内容，判断函数中是否确实存在漏洞，如果不符合第三步获取潜在漏洞信息时触发的条件，则删除潜在漏洞信息，否则将潜在漏洞信息更新为实际漏洞信息。
 5. 列举真实的代码漏洞点
- **STYLE:** 学术分析报告
- **TONE:** 专业，学术
- **AUDIENCE:** 任何了解代码安全，漏洞安全的人
- **RESPONSE EXAMPLE:** test “变量是一个无符号变量，数据流分析发现执行代码 “test += test2” (“test2 “也是一个无符号变量)，执行代码上下文的控制流分析发现，上面的代码没有对 “test “变量进行作用域检查，执行代码下面更新的 “test “变量也没有溢出检查，因此存在实际漏洞。
- **START ANALYSIS:** 开始分析

有效输出：

1. Potential Integer Overflow with RouterLifetime, ReachableTime, and **ParsedLength**:
 - **Description:** RouterLifetime and ReachableTime are obtained via RtlUlongByteSwap and can result in integer overflow if the input value is too large. **Similarly, ParsedLength is continuously incremented without bounds checks.**
 - **Code Locations:**

```
USHORT ParsedLength;
ULONG RouterLifetime, ReachableTime;
/**/
RouterLifetime = RtlUshortByteSwap(Advertisement->nd_ra_router_lifetime);
ReachableTime = RtlUlongByteSwap(Advertisement->nd_ra_reachable);
ParsedLength += Length;
```
 - **Mitigation:** Add checks to ensure values do not exceed expected ranges:

```
if (RouterLifetime > MAX_ROUTER_LIFETIME) {
    RouterLifetime = MAX_ROUTER_LIFETIME;
}
```
 - **问题:** The RtlUshortByteSwap routine reverses the ordering of the two bytes in a 16-bit unsigned integer value; The RtlUlongByteSwap routine reverses the ordering of the four bytes in a 32-bit unsigned integer value. 针对 RouterLifetime 和 ReachableTime 变量，如果传入参数符合条件，那么返回值也符合条件

2. Possible Mismatched Data Lengths in NetioAdvanceNetBuffer and NetioRetreatNetBuffer:

- **Description:** The function advances and retreats NetBuffer using ParsedLength, which may result in inconsistencies if ParsedLength does not match actual data length.
- Code Locations:

```
NetioAdvanceNetBuffer(NetBuffer, ParsedLength); // add data buffer
// ...
NetioRetreatNetBuffer(NetBuffer, ParsedLength, 0); // free data buffer
```

- Mitigation: Ensure ParsedLength accurately reflects the data processed, potentially adding verification after buffer operations.
- 基本命中 CVE-2021, 函数 NetioAdvanceNetBuffer 与 NetioRetreatNetBuffer, 如果 ParsedLength 与实际数据长度不匹配, 可能会导致不一致。

3. Unchecked Return Value from NetioGetDataBuffer:

- Description: The return value of NetioGetDataBuffer for OptionPrefix and OptionRoute is not checked for NULL before accessing their members.
- Code Locations:

```
OptionPrefix = NetioGetDataBuffer(NetBuffer, Length, &OptionBufferPrefix, 1, 0);
OptionRoute = NetioGetDataBuffer(NetBuffer, Length, &OptionBufferRoute, 1, 0);
```

- Mitigation: Check if the return value of NetioGetDataBuffer is NULL before accessing members
- 问题: 通过 github 查找, 发现 [NdisGetDataBuffer function](#) 返回值可能为 NULL¹

无效输出:

4. Buffer Overflow Risk with DlAddressBuffer:

- Description: The function assigns data to DlAddressBuffer with a maximum length defined by DL_ADDRESS_LENGTH_MAXIMUM. There is a risk that if DL_ADDRESS_LENGTH_MAXIMUM is smaller than the length of the data retrieved by NetioGetDataBuffer, it could cause a buffer overflow.
- Code Location:

```
UCHAR DlAddressBuffer[DL_ADDRESS_LENGTH_MAXIMUM], *DlAddress = NULL;
/**/
```

```
DlAddress = NetioGetDataBuffer(NetBuffer, Length, DlAddressBuffer, 1, 0);
```

- Mitigation: Ensure that DL_ADDRESS_LENGTH_MAXIMUM² is large enough to accommodate the maximum possible data length or add proper checks before copying data into DlAddressBuffer.
- 无效原因: 通过 github 查找, 发现 DL_ADDRESS_LENGTH_MAXIMUM 变量定义为 32, 难触发

5. Potential Wild Pointer Dereference:

- The Neighbor pointer, after being assigned by IppFindOrCreateNeighborUnderLock, is used without being checked for a NULL value before dereferencing in the statement Neighbor->IsRouter = TRUE;.
- Code Location:

```
Neighbor = IppFindOrCreateNeighborUnderLock(Interface, \
    Args->SourceSubInterface, (const UCHAR *) RemoteAddress, NlatUnicast);
```

- Mitigation: Ensure Neighbor is checked for NULL before dereferencing it
- 无效原因: 存在异常控制流代码, 未检测到 `if (Neighbor != NULL) {/**/}`, 改进
- 改进: 进行漏洞点可利用性排序, 有概率进行剔除

¹ #define NetioGetDataBuffer NdisGetDataBuffer 代码参考

² pub const DL_ADDRESS_LENGTH_MAXIMUM: u32 = 32u32; 代码参考

参考文献

1. Defect Detection-非 Transformer 结构的神经元方法（缺陷检测）

- Automatically learning semantic features for defect prediction (ICSE-2016) ¹
- VulDeePecker: A Deep Learning-Based System for Vulnerability Detection (NDSS-2018b) ²
- Cross-Project Transfer Representation Learning for Vulnerable Function Discovery (TII-2018) ³
- DeepBugs: a learning approach to name-based bug detection (OOPSLA-2018) ⁴
- Automated Vulnerability Detection in Source Code Using Deep Representation Learning (ICMLA-2018) ⁵
- SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities (TDSC-2022) ⁶
- Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks (NeurIPS-2019) ⁷
- Improving bug detection via context-based code representation learning and attention-based neural networks (OOPSLA-2019) ⁸
- Vulnerability detection with fine-grained interpretations (FSE-2021) ⁹
- **VulDeeLocator: A Deep Learning-Based Fine-Grained Vulnerability Detector (TDSC-2021)** ¹⁰-程序切片技术
- μ VulDeePecker: A Deep Learning-Based System for Multiclass Vulnerability Detection (TDSC-2019) ¹¹
- Deep Learning Based Vulnerability Detection: Are We There Yet? (TSE-2021) ¹²
- Self-Supervised Bug Detection and Repair (NeurIPS-2021) ¹³
- Vulnerability detection with fine-grained interpretations (FSE-2021) ¹⁴
- ReGVD: revisiting graph neural networks for vulnerability detection (ICSE-2022) ¹⁵

1 <https://doi.org/10.1145/2884781.2884804>

2 <https://doi.org/10.1109/TDSC.2019.2942930>

3 <https://doi.org/10.1109/TII.2018.2821768>

4 <https://doi.org/10.1145/3276517>

5 <https://doi.org/10.1109/ICMLA.2018.00120>

6 <https://doi.org/10.1109/TDSC.2021.3051525>

7 https://proceedings.neurips.cc/paper_files/paper/2019/file/49265d2447bc3bbfe9e76306ce40a31f-Paper.pdf

8 <https://doi.org/10.1145/3360588>

9 <https://doi.org/10.1145/3468264.3468597>

10 <https://doi.org/10.1109/TDSC.2021.3076142>

11 <https://doi.org/10.1109/TDSC.2019.2942930>

12 <https://doi.org/10.1109/TSE.2021.3087402>

13 https://proceedings.neurips.cc/paper_files/paper/2021/file/ea96efc03b9a050d895110db8c4af057-Paper.pdf

14 <https://doi.org/10.1145/3468264.3468597>

15 <https://doi.org/10.1145/3510454.3516865>

1. Defect Detection-Transformer 结构的神经元方法（缺陷检测）

- ~~VulBERTa: Simplified Source Code Pre-Training for Vulnerability Detection (IJCNN-2022)~~¹—深度学习只给出分类
- LineVul: a transformer-based line-level vulnerability prediction (MSR-2022)²—代码行/函数级别漏洞检测(深度学习架构)—**BASELINE**
- ~~Transformer-based Vulnerability Detection in Code at EditTime: Zero-shot, Few-shot, or Fine-tuning?~~ (arxiv)³—代码编辑时进行动态漏洞检测—CodeQL 检测到的漏洞数据进行训练
- ~~Evaluating Instruction-Tuned Large Language Models on Code Comprehension and Generation~~ (arxiv)⁴—大模型代码理解与代码生成
- **Prompt-Enhanced Software Vulnerability Detection Using ChatGPT (ICSE-2024)**⁵—Chatgpt 提示词驱动的漏洞检测方法
- ~~Towards Causal Deep Learning for Vulnerability Detection~~ (arxiv)⁶—深度学习的工作
- **How Far Have We Gone in Vulnerability Detection Using Large Language Models** (arxiv)⁷—LLM 在漏洞检测的调研-GPT4 最好
- ~~LLMs Cannot Reliably Identify and Reason About Security Vulnerabilities (Yet?): A Comprehensive Evaluation, Framework, and Benchmarks~~ (arxiv)⁸—LLMs 漏洞评估框架
- **LLM4Vuln: A Unified Evaluation Framework for Decoupling and Enhancing LLMs' Vulnerability Reasoning** (arxiv)⁹—提出 LLMs 漏洞安全评估框架，找到零日漏洞
- ~~Multi-role Consensus through LLMs Discussions for Vulnerability Detection~~ (arxiv)¹⁰—利用 LLM 模拟代码审计-分类任务
- ~~A Comprehensive Study of the Capabilities of Large Language Models for Vulnerability Detection~~ (arxiv)¹¹—给出了可参考的提示词—GPT4 Basic (zero-shot) prompting 表现最好

2. Reference 可参考文献

- Unifying the Perspectives of NLP and Software Engineering: A Survey on Language Models for Code (arxiv)¹²
- LineVul: a transformer-based line-level vulnerability prediction (MSR-2022)² **BASELINE**
- How Far Have We Gone in Vulnerability Detection Using Large Language Models (arxiv)⁷ **GPT4 best**
- A Comprehensive Study of the Capabilities of Large Language Models for Vulnerability Detection (arxiv)¹¹ **GPT 提示词方法-自然语言即可**
- Prompt-Enhanced Software Vulnerability Detection Using ChatGPT (ICSE-2024)⁵ **图+数据流+符号执行分析**

1 <https://doi.org/10.1109/IJCNN55064.2022.9892280>

2 <https://doi.org/10.1145/3524842.3528452>

3 <https://doi.org/10.48550/arXiv.2306.01754>

4 <https://doi.org/10.48550/arXiv.2308.01240>

5 <https://doi.org/10.48550/arXiv.2308.12697>

6 <https://doi.org/10.1145/3597503.3639170>

7 <https://doi.org/10.48550/arXiv.2311.12420>

8 <https://doi.org/10.48550/arXiv.2312.12575>

9 <https://doi.org/10.48550/arXiv.2401.16185>

10 <https://doi.org/10.48550/arXiv.2403.14274>

11 <https://doi.org/10.48550/arXiv.2403.17218>

12 <https://doi.org/10.48550/arXiv.2311.07989>

代码理解与漏洞定位

1. 代码切片处理
2. 采用 Code Copilot 工具
3. 提示词 1: Read the above code and, to the best of your knowledge, explain to what operation the code “*****” corresponds to the actual RFCxxxx
4. 提示词 2: Please read the code again and based on what you have learned, is this code slice vulnerable and if so, and give a description of the most likely vulnerability that would trigger it (list only one)
5. 提示词 3: Read the optimized function names and their implementation frameworks inside the functions, and list all the network protocol RFC documents that you think are relevant (list the 3 most relevant and give the basis).
6. 提示词 4: Optimize code formatting for the above code (including variable names, code indentation, comment descriptions)
7. 提示词 5: Please check again for the vulnerabilities you have given, such as whether the variables involved in the vulnerability have been updated in the text prior to the point of vulnerability, whether the variables involved in the vulnerability are no longer used afterwards, or whether the occurrence of the vulnerability would have been defused or checked in some other way as well.

- 课题背景

1. 目标: Windows server 网络驱动
2. 攻击利用点: 开发人员未能完全遵循 RFC 规范 (规范更新, 没有强制要求)

- 解决的问题

1. 规范文字与代码对应 (工程实现与文档不一致),
2. (闭源, 没有源代码信息, 反汇编代码信息的缺失),

(RFC 文档分类框架, 编号定义理解??) (一致性问题验证) (Different testing, Fuzzing, 验证方法)

- 当前工作方法

1. 针对驱动文件进行逆向反汇编
2. 代码抽象语义获取: 针对反汇编的函数代码, 补充函数代码信息 (函数所在文件, 文件作用), 生成代码分析信息, 生成代码总结 (包含什么内容)
 1. 函数名+文件名->函数总结-函数代码缺少语义信息-消融实验
 2. 匹配问题 ($m * n$), 用小型人工标注数据集 (test) 去校验- (3 个独立+讨论+统一)
 3. 程序+ RFC 文档 -> 黑盒 -> 程序匹配哪些 RFC (实验)
 4. RFC 文档抽取/划分 (Hdiff: 基于 NLP 进行情感词的抽取, 基于目录, 内容进行切分, 样例说明) -> 规则生成
 5. 程序改进 (背景信息添加)

(挑战 1: 自动化/半自动方法, 形式化定, 闭源代码反汇编变量名语义)

1. 基于 LLM 的 RFC 文档映射: LLM 输入代码总结, 针对 RFC 文档进行遍历, 输出可能涉及的 RFC 文档编号
2. 基于人工的 RFC 文档映射细化: 人工进行 LLM 输出结果的校准, 将 LLM 输出的匹配结果进行检验, 而后缩小范围, 确定代码对应的 RFC 文档具体章节

- 样例说明

1. 获取到 Windows server 平台 tcpip.sys 驱动文件 (二进制程序), 网络没有源代码实现, 我想找到这个二进制程序的漏洞
2. 采用逆向工程的方法, 反汇编二进制程序, 获取到所有汇编函数共 5000 余个, 选取 CVE-2021-24074 漏洞函数 Ipv6pHandleRouterAdvertisement 作为样例说明
3. 针对漏洞函数代码, 补充函数代码背景信息 (属于 tcpip.sys 文件, tcpip.sys 文件功能及其所在位置), 利用 GPT4 TOOL (Code Copilot: 一个用特定提示词框架引导 LLM 专注于代码任务) 进行代码总结, 生成针对代码的抽象语义信息描述 (The function Ipv6pHandleRouterAdvertisement processes IPv6 router advertisement messages in a Windows driver. Here's a high-level overview of the function: Initialize Variables, Verify Message Validity..)
4. 将输出的 Function Code Overview 作为文本输入到 GPT4 TOOL (RFCGpt: 一个用特定提示词框架引导 LLM 专注于 RFC 文档解释或搜索), 输出匹配的 RFC 文档编号 (RFC4861, RFC6106, RFC4191, RFC8106, RFC6275), 出现漏报, 误报的情况。

- 当前工作方法存在的问题:

1. 无法确保 LLM 输出的结果是全面且准确的, 存在漏报与误报问题。
 - 漏报问题: 针对函数 Ipv6pHandleRouterAdvertisement, LLM 可能首先可以知道这个函数涉及 IPv6, RouterAdvertisement 路由广播消息类型, 而后按照这两个关键词去检索。然而这个函数漏洞的问题出现在 IPv6 选项的一个新特性 Jumogram, 该特性并未在 IPv6

中涉及，而是作为新的文档进行补充，因此 LLM 不会输出该 RFC，从而造成该漏洞无法被发现。不漏报会导致误报。

- 误报问题，针对该函数涉及的所有协议关键词进行匹配，达到减少漏报的目标。
`Ipv6pHandleRouterAdvertisement` 函数中的 `RouterAdvertisement` 类型报文可能涉及多个协议，例如 ICMPV6, DNS, Mobility 等。实际查看函数，发现函数只是与 ICMPV6 相关。

2. Hdifff 的思想：利用 HTML 相关的 RFC (7230-7235) 书写 HTML 相关的关键词框架 (HTML 协议涉及到的高频关键词以及协议构成) 进行匹配。利用 LLM 针对不同的协议去书写这样的框架。

- 第一篇大论文
 1. 背景：开发人员未能完全遵循 RFC 规范（规范更新，没有强制要求）
 2. 问题：闭源软件函数与 RFC 文档片段的匹配问题
 1. 代码工程实现与 RFC 文档自然语言说明存在语义差距，匹配难
 2. 代码层次：
 1. 开源代码可以利用源代码进行 RFC 文档的切片匹配-只需 RFC 文档切片处理
 2. 闭源软件无法获得源码，反汇编软件获取的反汇编代码会丢失源码信息，需要同时对代码和 RFC 文档进行处理
 3. 解决方法：
 1. 输入 1：RFC 文档
 1. 内容：一个协议相关的，尚未废除的 RFC 文档切片
 2. 格式：切分的 RFC 规则
 2. 输入 2：代码处理
 1. 范围：一个驱动反汇编的代码（函数级别）+ 背景信息
 - 函数所在的驱动，函数所在驱动的文件功能与位置
 - 汇编处理：
 2. 格式：
 3. 输出 3：匹配结果，代码与文本的相似度匹配问题
 4. 实验：
 1. 校验实验：构建一个人工标注的数据集验证（3 个独立+讨论+统一）
 2. 消融实验：
 1. 代码处理/代码不处理
 2. RFC 处理（人分段）/RFC 处理（小标题）/RFC 不处理
- 第二篇论文
 1. 背景：开发人员未能完全遵循 RFC 规范（规范更新，没有强制要求）
 2. 问题：
 1. 反汇编代码和 RFC 文档匹配问题是否一致的问题