00000000	00	0c	29	3e	02	d1	00	15	5d	be	bc	00	86	dd	60	00
00000010	00	00	00	00	00	ff	fe	80	00	00	00	00	00	00	db	90
00000020	74	8e	fc	5f	e6	2f	fe	80	00	00	00	00	00	00	ac	c6
00000030	51	28	79	2d	50	05	2b	00	c2	04	00	01	10	10	2b	ff
00000040	00	00	06	d5	00	15	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
....	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
00000830	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	2b	ff
....	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
00001030	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	2b	ff
....	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
00001830	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	2b	ff
....	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
00002030	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	2b	ff
....	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
....	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
0000F830	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	2c	ff
....	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
00010030	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	3a	00
00010040	00	01	56	56	56	56	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
....	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
00011040	xx	xx	xx	xx	xx	xx										

- IPv6 Message Address Index (16 bytes one line)
- Destination Mac Address in Ethernet II Header(6 bytes)
- Source Mac Address in Ethernet II Header(6 bytes)
- IPv6 type in Ethernet II Header(2 bytes)
- IP version,Traffic Class and Flow Label in IPv6 Header(4 bytes)
- Payload Length in IPv6 Header(2 bytes)
- Next header(IPv6 Hop-by-Hop Option) in IPv6 Header(1 byte)
- Hop Limit in IPv6 Header(1 byte)
- Source Ip Address in IPv6 Header(16 bytes)
- Destination Ip Address in IPv6 Header(16 bytes)
- Next header(Routing Header Option) in IPv6 Header(1 byte)
- Hop-by-Hop Options Header Length(1 byte)
- Jumbo Option Type(1 byte)
- Jumbo Option Length(1 byte)
- Jumbo Payload Length(4 bytes)
- Next header(Routing Header Option) in IPv6 Header(1 byte)
- Routing header Option Length(1 byte)
- Routing Type in Routing Header(1 byte)
- Segments Left in Routing Header(1 byte)
- Reserved Data in Routing Header(4 bytes)
- Next header(Fragment Header Option) in IPv6 Header(1 byte)
- Next header(ICMP for IPv6) in IPv6 Header(1 byte)
- Reserved in Fragment Header(1 byte)
- Fragment Offset + Res + M flag in Fragment Header(2 bytes)
- Indentification in Fragment Header(4 bytes)
- Random Fill Data

Figure 1: A single ICMPv6 packet with Jumbo option whose length is bigger than 65535 triggering CVE-2021-24074.

3.2 Case study 1: CVE-2021-24074

Network can be divided into seven layers, physical layer, data link layer, network layer, transport layer, session layer, presentation layer, and application layer, according to Open System Interconnection/Reference Model. The network layer enables seamless data transmission between two end systems, including addressing, routing, and connection management. IP (Internet Protocol) protocol is the core component of this layer, implemented in Windows by the C:\Windows\System32\drivers\tcpip.sys driver. IPv4 addresses are 32 bits long but face exhaustion. IPv6 solves this with 128-bit addresses.

A major improvement in IPv6 is the introduction of extension headers, providing optional internet-layer information. These headers follow the IPv6 base header and include Hop-by-Hop Options, Routing, and Fragment Headers, enhancing IPv6's flexibility and adaptability.

For IPv6 messages exceeding 65,535 bytes, the Jumbo payload option in the Hop-by-Hop Options Header can be used, which is particularly useful for large data transfers. The

```
1  __int64 __fastcall Ipv6pHandleRouterAdvertisement(//..args..//){
2    // <1> NetBuffer
3    struct _NET_BUFFER *v9;
4    // <2> Option Total Length (max 65535)
5    unsigned __int16 v21;
6    // <3> Option pointer
7    KIRQL *v28;
8    // <4> Option Length
9    unsigned __int16 v29;
10   // ... Validate the Router Advertisement ...
11   while ( 1 ) {
12     // <4−1> Get current Option pointer and current Option Length
13     v28 = (KIRQL *)NdisGetDataBuffer(v9, 2u, &v231, 1u, 0);
14     v29 = 8 * v28[1];
15     // ...
16     // Move forward to the next option. (1)
17     // Keep track of the Parsed Length, so we can use it below to back up.
18     NdisAdvanceNetBufferDataStart(v9, v29, 0, 0);
19     // ...
20     // (2) <2−1> Option Total Length Update WITH OVERFLOW RISK
21     v21 += v29;/*..............................................v21 VARIABLE OVERFLOW*/
22   }
23   // ...
24   NdisRetreatNetBufferDataStart(v9, v21, 0, NetioAllocateMdl);
25   // ...
```

Listing 1: CVE-2021-24074 Vulnerable Code Snippet. The variable `v21`, which records the total length of the Option, is initialized as a short integer variable with an upper limit of 65536 at line 5. In the while loop structure, the variable `v21` is continuously incremented by the variable `v29`, which records the current Option length, at line 21. Since no checks are performed, there is a risk of integer overflow.

Jumbo payload length allows multiple extension header options in the IPv6 header. Each option includes type, length, and data fields, forming a dynamically extensible list structure, increasing protocol flexibility.

However, the list design has drawbacks. The main issue is the lack of an upper limit on the number of options, potentially causing integer overflow when using a fixed-size variable to record the total length of the option chain. This can lead to data processing errors and potential vulnerabilities.

CVE-2021-24074 is an integer overflow vulnerability found in the `Ipv6pHandleRouterAdvertisement` function of the Windows Operating System TCP/IP driver, which processes Router Advertisement message with Jumbo payload option, posing a significant security risk by allowing remote code execution.

Listing 1 shows the vulnerable code snippet reverse-engineered from the Windows TCP/IP driver by reverse tool IDApro. While parsing the message IPv6 extensive header (Figure 1) options, the function establishes a variable called `ParsedLength` to keep track of the total length of the data stored in the option list. As the number of parsed valid options increases, the `ParsedLength` variable accumulates the lengths of valid option. However, since `ParsedLength` is of type `USHORT`, it can only hold values up to 65535, leading to a potential overflow.

00000000	ff	ff	ff	ff	ff	ff	00	15	5d	be	8b	01	08	06	xx	xx
00000010	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
00000020	xx	xx														

ARP Message Address Index (16 bytes one line)
Destination Broadcast Address (6 bytes)
Source Mac Address (6 bytes)
ARP type (2 bytes)
Kernel Address Information Leakage(20 bytes)

Figure 2: A single ARP packet construction whose useful info length is only 15 bytes triggering CVE-2022-30223.

3.3 Case study 2: CVE-2022-30223

Virtual Network Interface Cards (vNICs) are essential in virtualized environments, allowing virtual machines (VMs) to communicate over a network as if they were physical devices. vNICs replicate the functionality of physical Network Interface Cards (NICs) but are implemented through software, providing greater flexibility and scalability. They facilitate network connectivity for VMs, enabling them to send and receive data over internal and external networks seamlessly. This connectivity ensures VMs operate effectively within a virtual infrastructure.

vNICs also allow the hypervisor to dynamically manage and allocate network resources, optimizing network performance and usage. This dynamic management is crucial for maintaining an efficient virtual environment. Additionally, vNICs enhance isolation and security by ensuring VMs are isolated from each other, preventing unauthorized access and potential security breaches. `vmswitch.sys` is a vital component of the Hyper-V platform in Windows, responsible for managing virtual network switches, ensuring efficient and secure network traffic routing between virtual machines.

TCP/UDP checksum offload in Windows improves network performance by offloading checksum calculations from the CPU to the network adapter, reducing CPU load and enhancing throughput and latency. However, it can cause compatibility issues with certain configurations or older hardware.

In a Linux VM and Windows host scenario, the interplay between Net Buffer List (NBL), RNDIS message, and skb message facilitates seamless cross-platform data transmission. When a Linux VM sends a DNS UDP broadcast packet, it is encapsulated within an skb message, which is then passed to the virtual network interface and transmitted through the vmbus to the Windows host. The vmbus driver on the Windows host decapsulates the packet, converts it into an NBL, and then encapsulates it into an RNDIS message. The Windows networking stack processes the RNDIS message and handles the DNS UDP broadcast packet.

Conversely, when the Windows host sends an ARP reply packet, it is encapsulated within an NBL, converted into an RNDIS message, and transmitted through the vmbus to the Linux VM. The Linux VM decapsulates the RNDIS message into an skb message, which is then processed by the Linux networking stack.

As shown in Figure 2, a critical information leak vulnerability arises when the hardware does not support TCP/UDP checksum or it is disabled. The flawed validation compares the length of the skb message instead of the RNDIS message, leading to excessive data copying and unintended information leakage. This vulnerability can be triggered when the Linux VM sends a DNS UDP broadcast packet or the Windows host sends an ARP reply packet, causing sensitive memory content to be improperly transmitted.

```
__int64 __fastcall VmsNblHelperCreateCloneNbl(//..args..//){
  // <1> Bytes to copy
  int v24, v60;
  // <2> CheckSum bit
  char v26;
  // ...
  // <2−1> CheckSum bit get
  v23 = SrcNetBufferList−>NetBufferListInfo[0];
  v26 = v23 != 0i64 ? a6 : 0;
  if ( v26 ) {
    // TcpCheckSum bit check Not Pass
    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, 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);
        // ...
```

Listing 2: CVE-2022-30223 Vulnerable Code Snippet. The vulnerability stems from inconsistent value assignment. While `v24` is set to 34 bytes, `v60` might retain a larger value if certain code paths are skipped. When `v60` is used to determine how much data to copy, it could lead to copying more than intended. This excess data may contain sensitive information from adjacent memory, resulting in an information leak.

Listing 2 demonstrates the use of a constant value for the byte count during a netbuffer copy operation, labeled as CVE-2022-30223, found in the `VmsNblHelperCreateCloneNbl` function reverse engineered from the `vmswith.sys` file. Relying on a hardcoded value instead of dynamically checking the message length leads to a significant security flaw, as additional data beyond the intended message is copied and transmitted, potentially exposing sensitive kernel memory information.

4 Related Work

Binary Code Vulnerability Detection. Jayakrishna et al. [2] present ARBITER, a groundbreaking framework that seamlessly combines static analysis and dynamic symbolic execution (DSE) to identify vulnerabilities in binary programs without the need for source code. This hybrid approach significantly enhances both precision and scalability, allowing for the successful detection of complex vulnerabilities such as integer overflows and unchecked return values. In a large-scale evaluation involving 76,516 binaries, ARBITER identified five new vulnerabilities.

He et al. introduces a novel binary code representation called Semantics-Oriented Graph (SOG) that reveals complete semantic structures of binary code while discarding semantically independent information. Additionally, it designs a multi-head softmax aggregator that effectively aggregates multiple aspects of the SOG, significantly enhancing system performance. The implementation of HermesSim, based on the SOG representation and the multi-head softmax aggregator, is presented as an efficient solution for binary code similarity detection. Extensive experiments were conducted to demonstrate the effectiveness of SOG over previous binary code representations and the superior performance of HermesSim over state-of-the-art methods. In real-world validation, HermesSim successfully identified 5 CVEs and 10 related vulnerable functions in RTOS firmware images from three vendors (TP-Link, Mercury, and Fast). This robust performance indicates HermesSim's strong capability in capturing zero-day vulnerabilities, significantly improving the efficiency and accuracy of binary code similarity detection and real-world vulnerability identification compared to existing methods.

LLMs on Vulnerability Detection. Fang et al. [1] introduced HPTSA (Hierarchical Planning and Task-Specific Agents), a multi-agent framework which significantly enhances the capability to autonomously exploit zero-day vulnerabilities and is validated through rigorous experiments. The introduction of task-specific agents, tailored to exploit particular types of vulnerabilities, combined with the hierarchical planning agent's strategic navigation and deployment, results in a more efficient and effective exploitation process. The use of relevant documents and tools further aids in understanding and attacking the system, ensuring that the task-specific agents are well-informed and capable.

The findings by Zhou et al. [5] highlight the feasibility and effectiveness of using LLMs for code understanding and vulnerability detection. Through effective pre-training, fine-tuning, and data augmentation techniques, LLMs can achieve high accuracy and robustness, making them invaluable tools for improving software security and reliability.

Zhang et al.'s study [4] demonstrates the significant potential of LLMs like ChatGPT in software vulnerability detection when enhanced with well-designed prompts and auxiliary information. The study utilized role-based prompts, which reduce biases and increase accuracy, and chain-of-thought prompting [3], which enhances the model's understanding of complex code. The integration of auxiliary information, such as data flow graphs (DFGs) and API call sequences, significantly improved detection accuracy, achieving an average improvement of up to 16%. These methods demonstrated high accuracy in detecting grammar-related and boundary-related vulnerabilities, making LLMs invaluable tools for complex code analysis and software security.

References

[1] Richard Fang, Rohan Bindu, Akul Gupta, Qiusi Zhan, and Daniel Kang. Teams of llm agents can exploit zero-day vulnerabilities, 2024.

[2] Jayakrishna Vadayath, Moritz Eckert, Kyle Zeng, Nicolaas Weideman, Gokulkrishna Praveen Menon, Yanick Fratantonio, Davide Balzarotti, Adam Doupé, Tiffany Bao, Ruoyu Wang, Christophe Hauser, and Yan Shoshitaishvili. Arbiter: Bridging the static and dynamic divide in vulnerability discovery on binary programs. In 31st USENIX Security Symposium (USENIX Security 22), pages 413–430, Boston, MA, August 2022. USENIX Association.

[3] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, Advances in Neural Information Processing Systems, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.

[4] Chenyuan Zhang, Hao Liu, Jiutian Zeng, Kejing Yang, Yuhong Li, and Hui Li. Prompt-enhanced software vulnerability detection using chatgpt. In Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, ICSE-Companion '24, page 276–277, New York, NY, USA, 2024. Association for Computing Machinery.

[5] Xin Zhou, Sicong Cao, Xiaobing Sun, and David Lo. Large language model for vulnerability detection and repair: Literature review and the road ahead.