EMPEROR : ICMPv6 P2P communication without 3rd party

Nikolaos Tsapakis, George Tselos

Whoami

Nikolaos Tsapakis is a reverse engineering enthusiast and poetry lover from Greece. He is working as a Security Engineer. He has been writing papers or presented for Virus Bulletin, 2600, LeHack, Symantec, Hakin9, Athcon, DeepSec.

George Tselos is a computer science tutor who lives and works in Athens, Greece. He is interested in embedded systems, microcontrollers and peripheral device development.

The problem

- P2P communication needs a 3rd party service to initiate like a stun server which provides NAT traversal
- Routers and NATs in network infrastructure block direct access to ports exposed on the internet
- Example of services that use 3rd party server are Skype, Zoom, Viber
- Why having intermediates ?
- https://www.cyberyodha.org/2023/04/what-is-stun-protocol.html

The idea

- (RFC 4890) Recommendations for Filtering ICMPv6 Messages in Firewalls
- Traffic that must not be dropped
- Do manufacturers for routers and firewalls respect by default that RFC ?

ICMPv6

- The unique IPv6 address makes the need for NAT traversal obsolete
- Protocol does not use ports
- Enough space for the payload
- Python 3 support

Traffic that must not be dropped

Davies & Mohacsi Informational [Page 13]

RFC 4890 ICMPv6 Filtering Recommendations May 2007

4.3.1. Traffic That Must Not Be Dropped

Error messages that are essential to the establishment and maintenance of communications:

- o Destination Unreachable (Type 1) - All codes
- o Packet Too Big (Type 2)
- o Time Exceeded (Type 3) - Code 0 only
- o Parameter Problem (Type 4) - Codes 1 and 2 only

Message format (RFC)

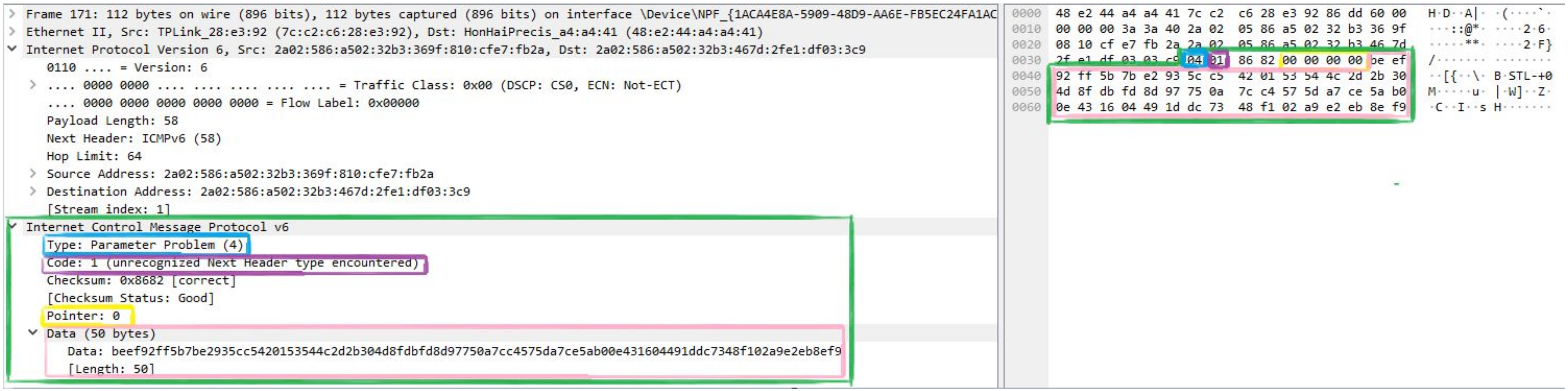
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Code      |          Checksum             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Pointer                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    As much of invoking packet                 |
+               as possible without the ICMPv6 packet           +
|               exceeding the minimum IPv6 MTU [IPv6]           |
```

```
IPv6 Fields:

Destination Address

               Copied from the Source Address field of the invoki
               packet.

ICMPv6 Fields:

Type           4

Code           0 - Erroneous header field encountered
               1 - Unrecognized Next Header type encountered
               2 - Unrecognized IPv6 option encountered

Pointer        Identifies the octet offset within the
               invoking packet where the error was detected.
```

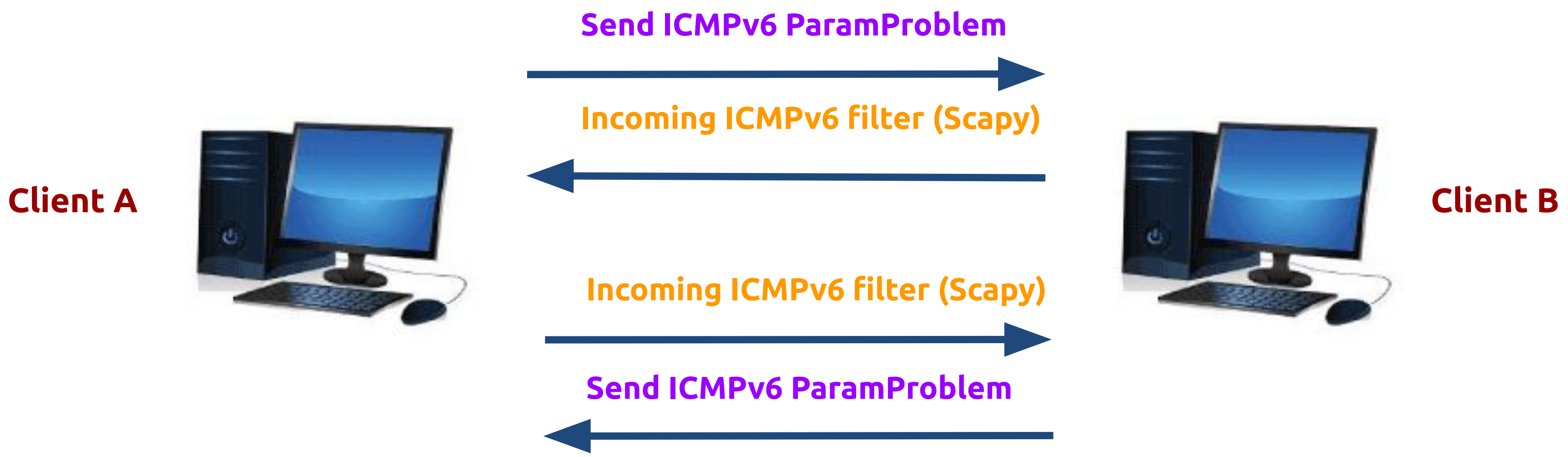
Message format (python)

```
# IPv6 packet
IPv6_icmp = IPv6()
IPv6_icmp.src = src
IPv6_icmp.dst = dst

# ICMPv6 following
IPv6_icmp.nh  = 58

# ICMPv6 ParamProblem message type
ICMPv6_ParamProblem       = ICMPv6ParamProblem()
ICMPv6_ParamProblem.type = 4
ICMPv6_ParamProblem.code = 1
ICMPv6_ParamProblem.ptr  = 0

# Set verbose to False to avoid printing information messages
packet = IPv6_icmp/ICMPv6_ParamProblem/Raw(load=data)
send(packet, count=1, verbose=False)
```

Message format (network)



P2P client


Client A
Send ICMPv6 ParamProblem
Incoming ICMPv6 filter (Scapy)
Incoming ICMPv6 filter (Scapy)
Send ICMPv6 ParamProblem
Client B

P2P client (sender)

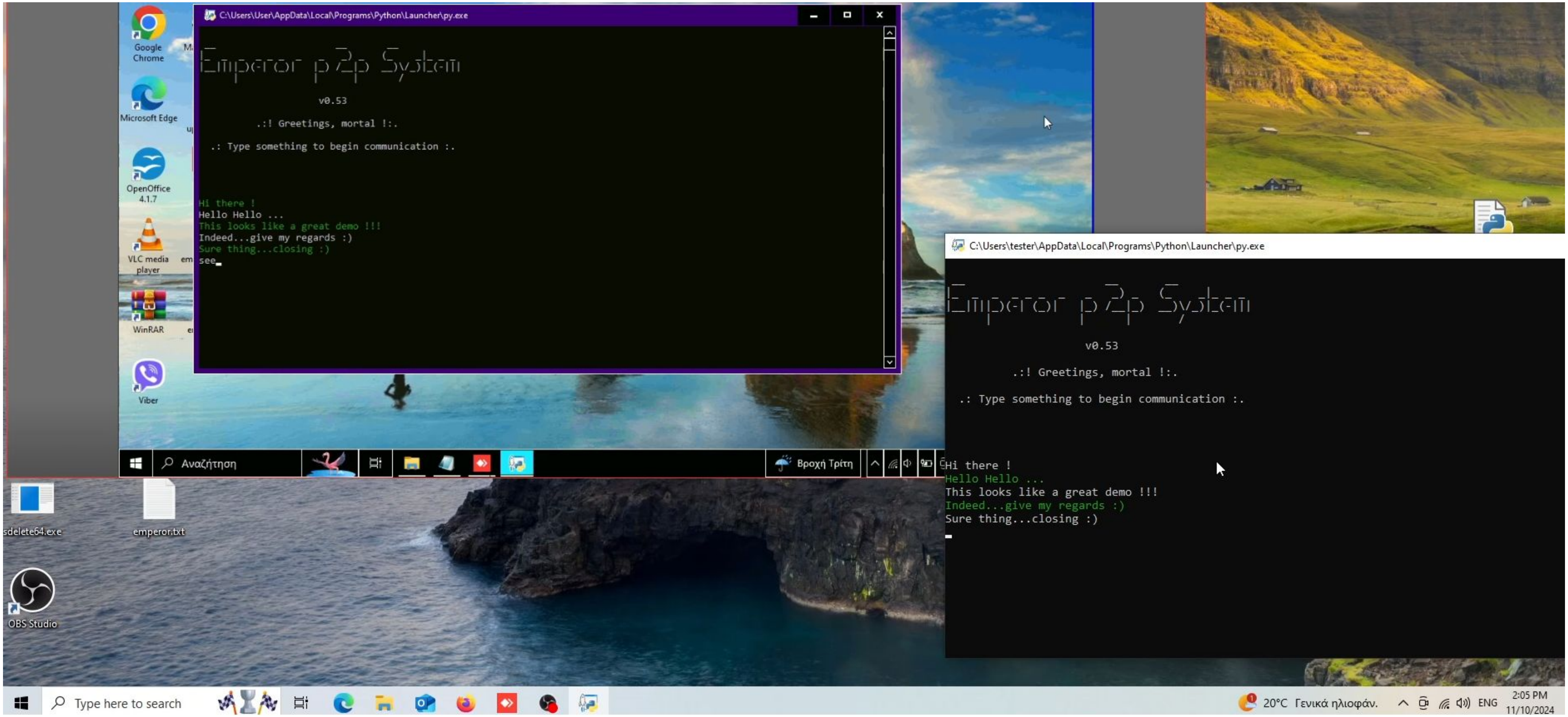
```python
def sender(src, dst, data):

    # IPv6 packet
    IPv6_icmp = IPv6()
    IPv6_icmp.src = src
    IPv6_icmp.dst = dst
    # IPv6_icmp.hlim = 1

    # ICMPv6 following
    IPv6_icmp.nh  = 58

    # ICMPv6 ParamProblem message type
    ICMPv6_ParamProblem      = ICMPv6ParamProblem()
    ICMPv6_ParamProblem.type = 4
    ICMPv6_ParamProblem.code = 1
    ICMPv6_ParamProblem.ptr  = 0

    # Set verbose to False to avoid printing information messages
    packet = IPv6_icmp/ICMPv6_ParamProblem/Raw(load=data)
    send(packet, count=1, verbose=False)
```

P2P client (receiver)

```
# pip install scapy
from scapy.all import *
```

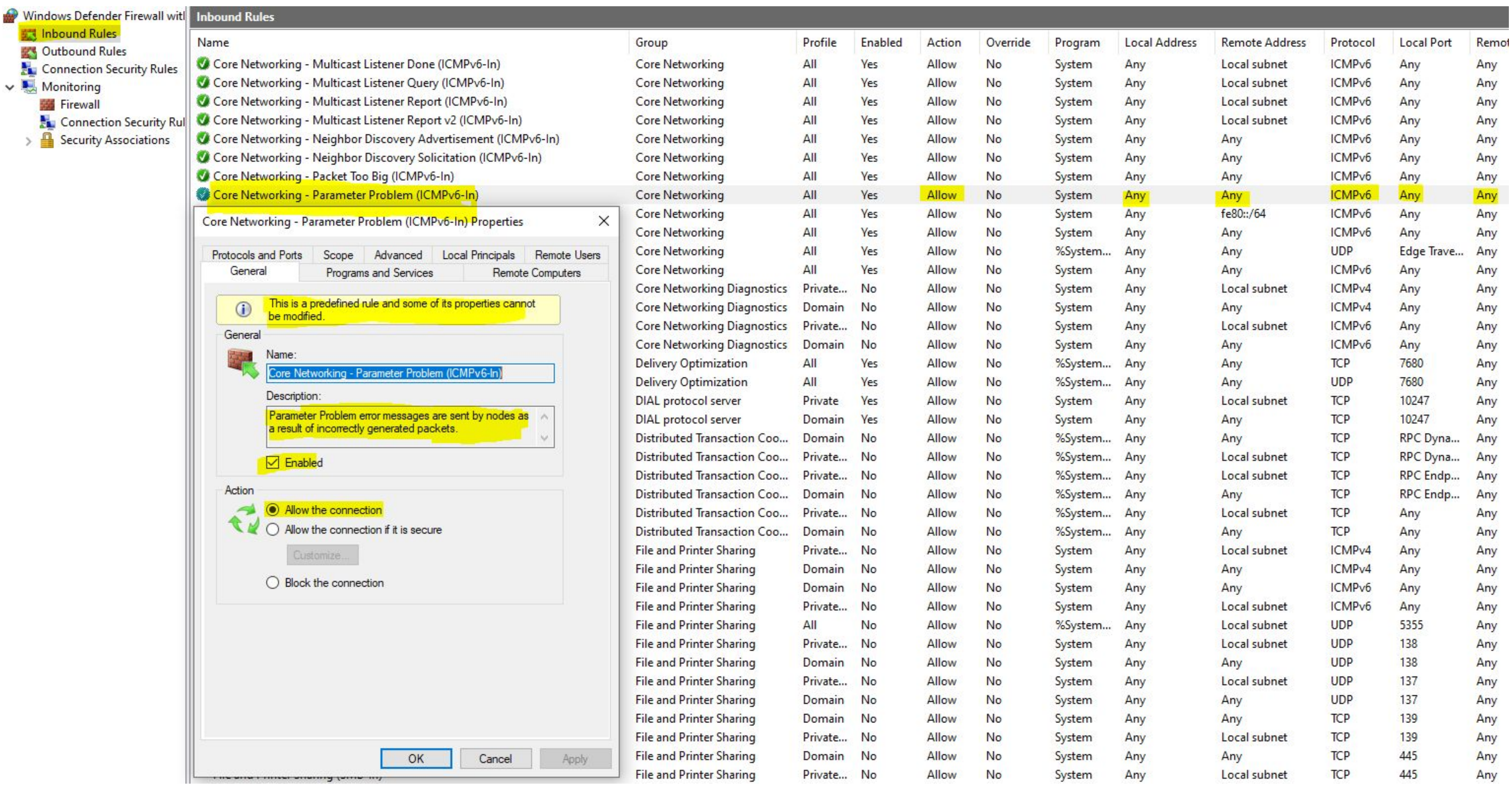
```
def receiver(dst, rc6, iv_data):
    while True:
        # Report only specific packet from specific ipv6 source address
        capture = sniff(filter="icmp6 && icmp6[0] = 4 && icmp6[1] = 1 && ip6 dst " + str(dst), count=1)
        packet = capture[0]
        packet = bytes(packet)
        # print(packet)
        data = packet[62:]
```

P2P client (demo)



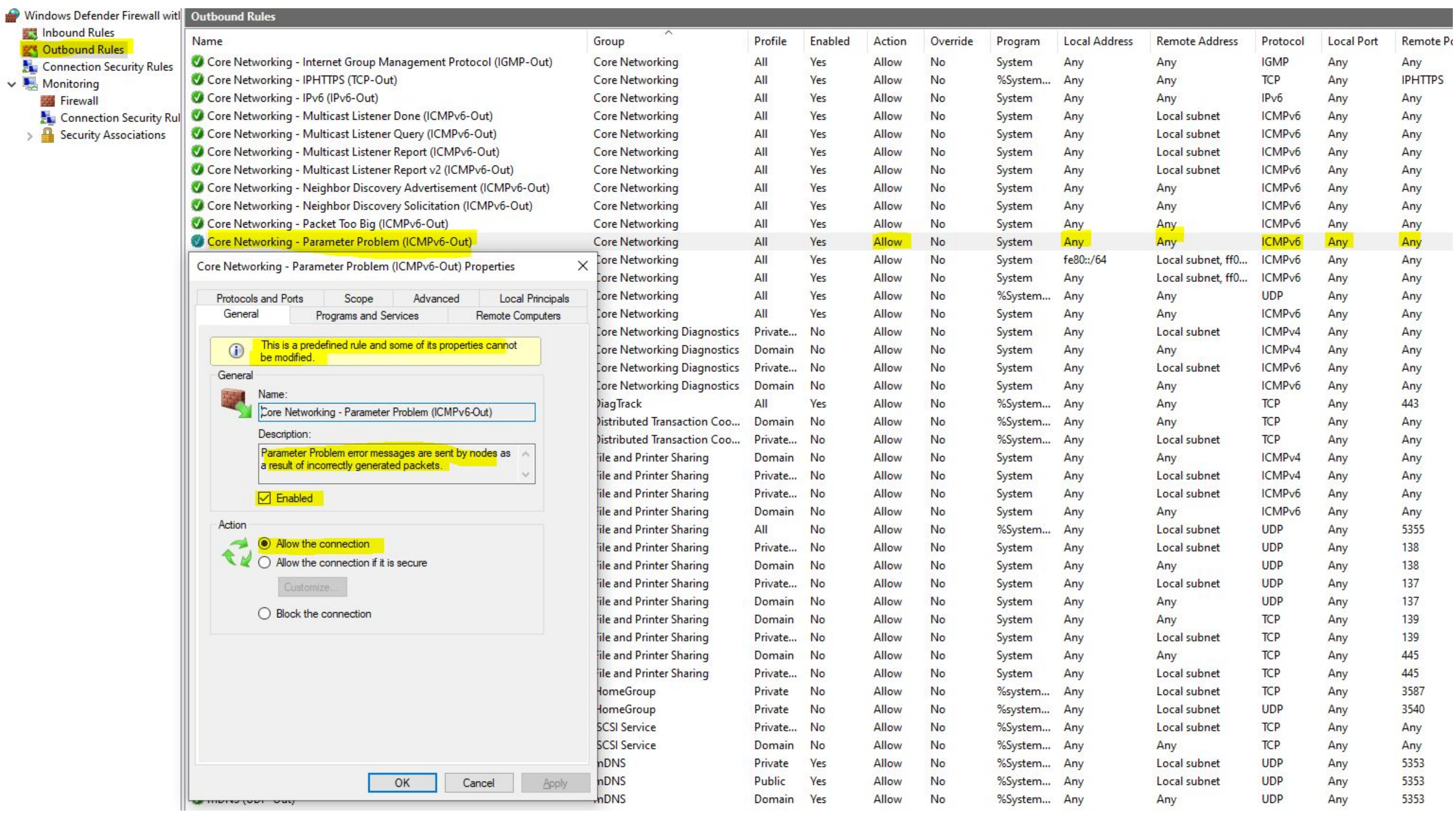
DEEPSEC

OS firewall (in)

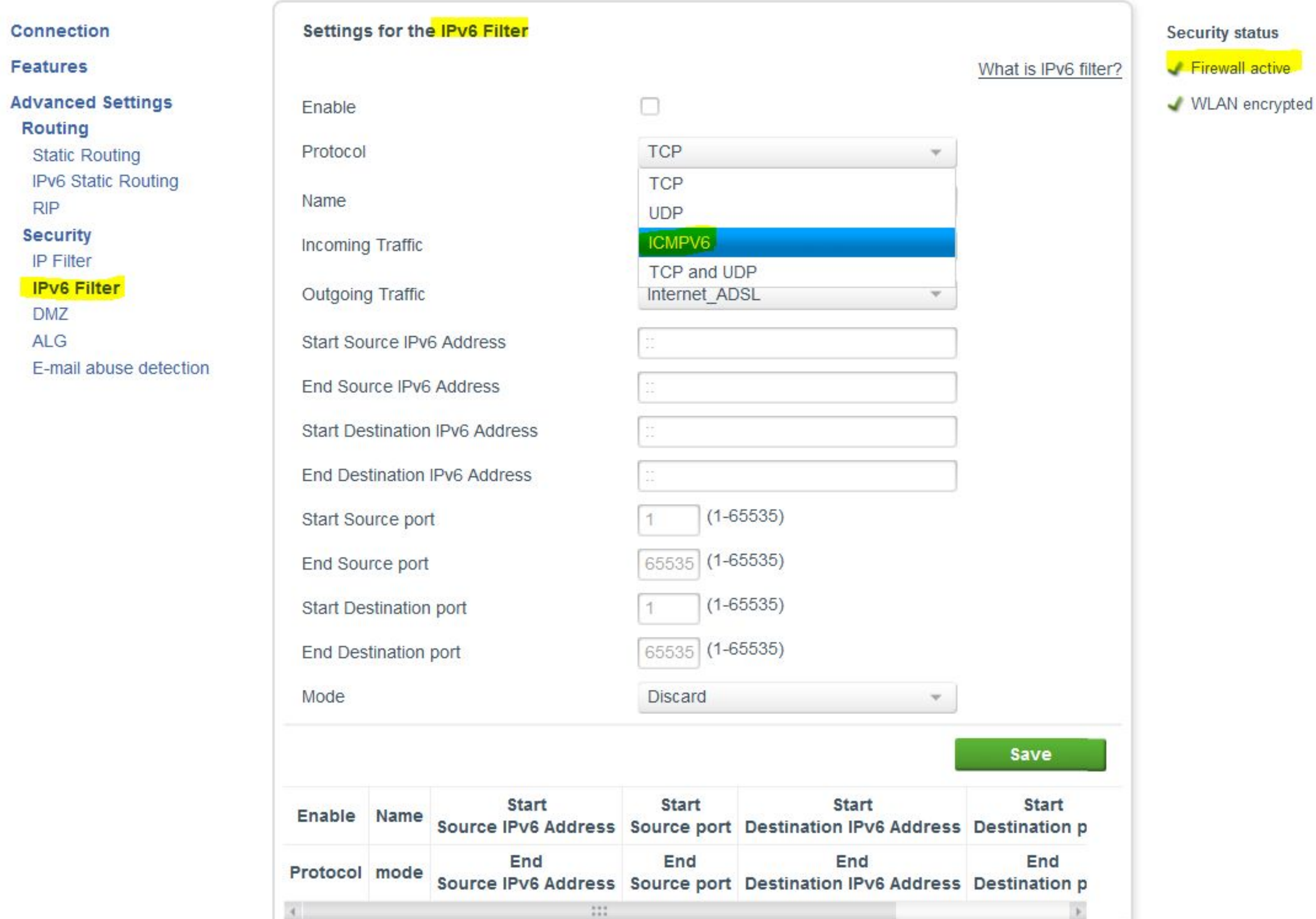


Name	Group	Profile	Enabled	Action	Override	Program	Local Address	Remote Address	Protocol	Local Port	Remot
Core Networking - Multicast Listener Done (ICMPv6-In)	Core Networking	All	Yes	Allow	No	System	Any	Local subnet	ICMPv6	Any	Any
Core Networking - Multicast Listener Query (ICMPv6-In)	Core Networking	All	Yes	Allow	No	System	Any	Local subnet	ICMPv6	Any	Any
Core Networking - Multicast Listener Report (ICMPv6-In)	Core Networking	All	Yes	Allow	No	System	Any	Local subnet	ICMPv6	Any	Any
Core Networking - Multicast Listener Report v2 (ICMPv6-In)	Core Networking	All	Yes	Allow	No	System	Any	Local subnet	ICMPv6	Any	Any
Core Networking - Neighbor Discovery Advertisement (ICMPv6-In)	Core Networking	All	Yes	Allow	No	System	Any	Any	ICMPv6	Any	Any
Core Networking - Neighbor Discovery Solicitation (ICMPv6-In)	Core Networking	All	Yes	Allow	No	System	Any	Any	ICMPv6	Any	Any
Core Networking - Packet Too Big (ICMPv6-In)	Core Networking	All	Yes	Allow	No	System	Any	Any	ICMPv6	Any	Any
Core Networking - Parameter Problem (ICMPv6-In)	Core Networking	All	Yes	Allow	No	System	Any	Any	ICMPv6	Any	Any
	Core Networking	All	Yes	Allow	No	System	Any	fe80::/64	ICMPv6	Any	Any
	Core Networking	All	Yes	Allow	No	System	Any	Any	ICMPv6	Any	Any
	Core Networking	All	Yes	Allow	No	%System...	Any	Any	UDP	Edge Trave...	Any
	Core Networking	All	Yes	Allow	No	System	Any	Any	ICMPv6	Any	Any
	Core Networking Diagnostics	Private...	No	Allow	No	System	Any	Local subnet	ICMPv4	Any	Any
	Core Networking Diagnostics	Domain	No	Allow	No	System	Any	Any	ICMPv4	Any	Any
	Core Networking Diagnostics	Private...	No	Allow	No	System	Any	Local subnet	ICMPv6	Any	Any
	Core Networking Diagnostics	Domain	No	Allow	No	System	Any	Any	ICMPv6	Any	Any
	Delivery Optimization	All	Yes	Allow	No	%System...	Any	Any	TCP	7680	Any
	Delivery Optimization	All	Yes	Allow	No	%System...	Any	Any	UDP	7680	Any
	DIAL protocol server	Private	Yes	Allow	No	System	Any	Local subnet	TCP	10247	Any
	DIAL protocol server	Domain	Yes	Allow	No	System	Any	Any	TCP	10247	Any
	Distributed Transaction Coo...	Domain	No	Allow	No	%System...	Any	Any	TCP	RPC Dyna...	Any
	Distributed Transaction Coo...	Private...	No	Allow	No	%System...	Any	Local subnet	TCP	RPC Dyna...	Any
	Distributed Transaction Coo...	Private...	No	Allow	No	%System...	Any	Local subnet	TCP	RPC Endp...	Any
	Distributed Transaction Coo...	Domain	No	Allow	No	%System...	Any	Any	TCP	RPC Endp...	Any
	Distributed Transaction Coo...	Private...	No	Allow	No	%System...	Any	Local subnet	TCP	Any	Any
	Distributed Transaction Coo...	Domain	No	Allow	No	%System...	Any	Any	TCP	Any	Any
	File and Printer Sharing	Private...	No	Allow	No	System	Any	Local subnet	ICMPv4	Any	Any
	File and Printer Sharing	Domain	No	Allow	No	System	Any	Any	ICMPv4	Any	Any
	File and Printer Sharing	Domain	No	Allow	No	System	Any	Any	ICMPv6	Any	Any
	File and Printer Sharing	Private...	No	Allow	No	System	Any	Local subnet	ICMPv6	Any	Any
	File and Printer Sharing	All	No	Allow	No	%System...	Any	Local subnet	UDP	5355	Any
	File and Printer Sharing	Private...	No	Allow	No	System	Any	Local subnet	UDP	138	Any
	File and Printer Sharing	Domain	No	Allow	No	System	Any	Any	UDP	138	Any
	File and Printer Sharing	Private...	No	Allow	No	System	Any	Local subnet	UDP	137	Any
	File and Printer Sharing	Domain	No	Allow	No	System	Any	Any	UDP	137	Any
	File and Printer Sharing	Domain	No	Allow	No	System	Any	Any	TCP	139	Any
	File and Printer Sharing	Private...	No	Allow	No	System	Any	Local subnet	TCP	139	Any
	File and Printer Sharing	Domain	No	Allow	No	System	Any	Any	TCP	445	Any
	File and Printer Sharing	Private...	No	Allow	No	System	Any	Local subnet	TCP	445	Any

OS firewall (out)



Router firewall



Client functions

- At least 1 destination IP address should be known
- Usual router/firewall setups would be bypassed
- RC6 encryption
- Peer communicates other Peer (P2P)
- No intermediate system assists in establishing communication

Future work

- P2P IP list sharing among Peers
- Tested on home routers and mobile phone hotspots
- RC6 key exchange using RSA
- Different Message Types

Demo

- Users on different geolocation behind different home routers
- Demo, presentation, paper, code (Open Source)
- https://github.com/nitsa

Q & A

Any questions ?

Thank you !