TRAINING SESSION NAME

Firewall

Table of contents

A. Main Content

1. Technical concept

1.1 scapy

1.1.1. Overview

- **Scapy** is a Python program that enables the user to send, sniff and dissect and forge network packets. This capability allows construction of tools that can probe, scan or attack networks.
- In other words, Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more.
- Scapy can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery. It can replace hping, arping... and even some parts of Nmap, tcpdump, and tshark.
- *In this topic, we will focus on Scapy with sending packet from windows to device for testing whether firewall is working properly or not.*

- *It can help us modify ethernet packet with these field:*

  - Ethernet (source/destination MAC address, Ethertype)

  - Vlan ID

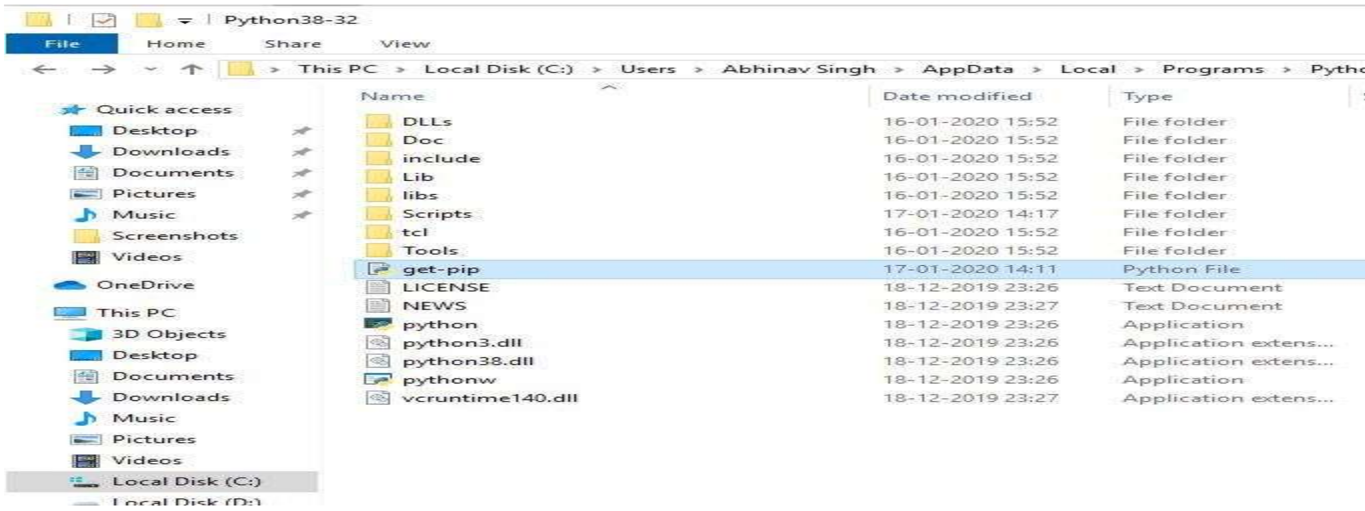
  - IPv4/Ipv6

  - protocol: TCP/UDP/ICMP...

1.1.2 install scapy on windows

- download and install latest version of python at https://www.python.org
- Install python-pip:

  - Download the get-pip.py (https://bootstrap.pypa.io/get-pip.py) file and store it in the same directory as python is installed.



Python38-32

File | Home | Share | View

This PC > Local Disk (C:) > Users > Abhinav Singh > AppData > Local > Programs > Pytho

Quick access: Desktop, Downloads, Documents, Pictures, Music, Screenshots, Videos
OneDrive
This PC: 3D Objects, Desktop, Documents, Downloads, Music, Pictures, Videos, Local Disk (C:), Local Disk (D:)

Name	Date modified	Type
DLLs	16-01-2020 15:52	File folder
Doc	16-01-2020 15:52	File folder
include	16-01-2020 15:52	File folder
Lib	16-01-2020 15:52	File folder
libs	16-01-2020 15:52	File folder
Scripts	17-01-2020 14:17	File folder
tcl	16-01-2020 15:52	File folder
Tools	16-01-2020 15:52	File folder
get-pip	17-01-2020 14:11	Python File
LICENSE	18-12-2019 23:26	Text Document
NEWS	18-12-2019 23:27	Text Document
python	18-12-2019 23:26	Application
python3.dll	18-12-2019 23:26	Application extens...
python38.dll	18-12-2019 23:26	Application extens...
pythonw	18-12-2019 23:26	Application
vcruntime140.dll	18-12-2019 23:27	Application extens...

  - Open cmd at this folder and run command to install python-pip:

    python get-pip.py

- After install python, open command prompt (cmd) and run follow commands to install some required packet:

  - *pip install scapy*

  - *pip install netaddr*

  - *pip install sqlalchemy*

1.2 tcpdump

1.2.1. Overview

- **tcpdump** is a data-network packet analyzer computer program that runs under a command line interface.
- User can use it to display TCP/IP and other packets being transmitted or received over a network to which the computer is attached.
- tcpdump is free software.
- Tcpdump works on most Unix-like operating systems: Linux, Solaris, FreeBSD, DragonFly BSD, NetBSD, OpenBSD, OpenWrt, macOS, HP-UX 11i, and AIX. In those systems. it uses libpcap for capturing packet.
- Other version of tcpdump for Windows is called WinDump; it uses WinPcap, the Windows version of libpcap.

1.2.2 Using tcpdump

a. We must check whether tcpdump was installed on device or not:

tcpdump --version

b. Show tcpdump help (refer to this page for more detail: https://www.tcpdump.org/manpages/tcpdump.1.html)

tcpdump -h

c. Some options that be usually used with tcpdump command in testing firewall:

Option		Example
-i *interface*	interface used to listen packet (ex: eth0, eth0.5, lo, any...)	
-Q *in|out|inout*	direction of packet (go out or in)	tcpdump -i eth0.5 -p udp -vv -Q in
-w *file*	write captured packets to a pcap file	tcpdump -i eth0.5 -w data.pcap
-r *file*	read a pcap file	tcpdump -r data.pcap
-p *protocol*	which protocol to be captured (tcp/udp/icmp/icmp6...) can be filter more than one protocol	tcpdump -i eth0.5 -p icmp6 -vv tcpdump -i eth0.5 -p udp or icmp6 -vv
-vv	show more information of captured packets	

d. Difference between some command:

command	**tcpdump -i eth0.5 -w data.pcap**	**tcpdump -i eth0.5**	**tcpdump -i eth0.5 -vv**
same	capture packet go in and out of interface eth0.5		
difference	write packets to file. data file can be read by command *tcpdump -r data.pcap* or copy to PC and read/analyze with wireshark	just display captured packet on command line	show more packets infor than **tcpdump -i eth0.5 command**

1.3 (Addition) Request IPv6 and setup for ping test

1.3.2 Config IPv6 and test ping

- Network Connection → Select Ethernet→ Properties→ Internet Protocol Version 6 (TCP/IPV6) Select Properties
- Select Use the following IPv6 address → Fill in the information below

IPv6 address: ***fd22:xxx:xxx:xxx::xx***
Subnet prefix length: ***64***

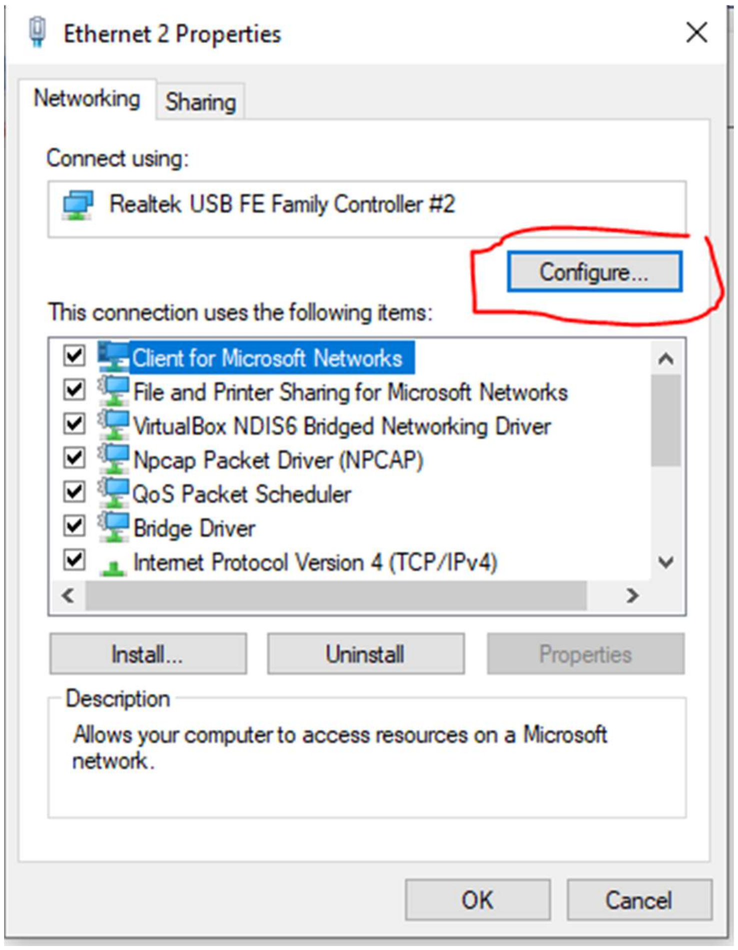
- Run Windows Command Prompt → Proceed with ping test as below:

Command: ping fd22:xxxx:xxx:5::14 -t

PC IP : fd22:xxxx:xxx:5::10
Device IP: fd22:xxxx:xxx:5::14

- If it does not work, access the CPU console window of the Device and enter the "ifconfig eth0" command to check if the IPv6 IP below is set.
- If not, configure IPv6 according to the instructions below:

Network Connection → Select Ethernet→ Properties→ Configure → Advanced > Set VLAN ID to 5


Ethernet 2 Properties

Networking | Sharing

Connect using:

Realtek USB FE Family Controller #2

Configure...

This connection uses the following items:

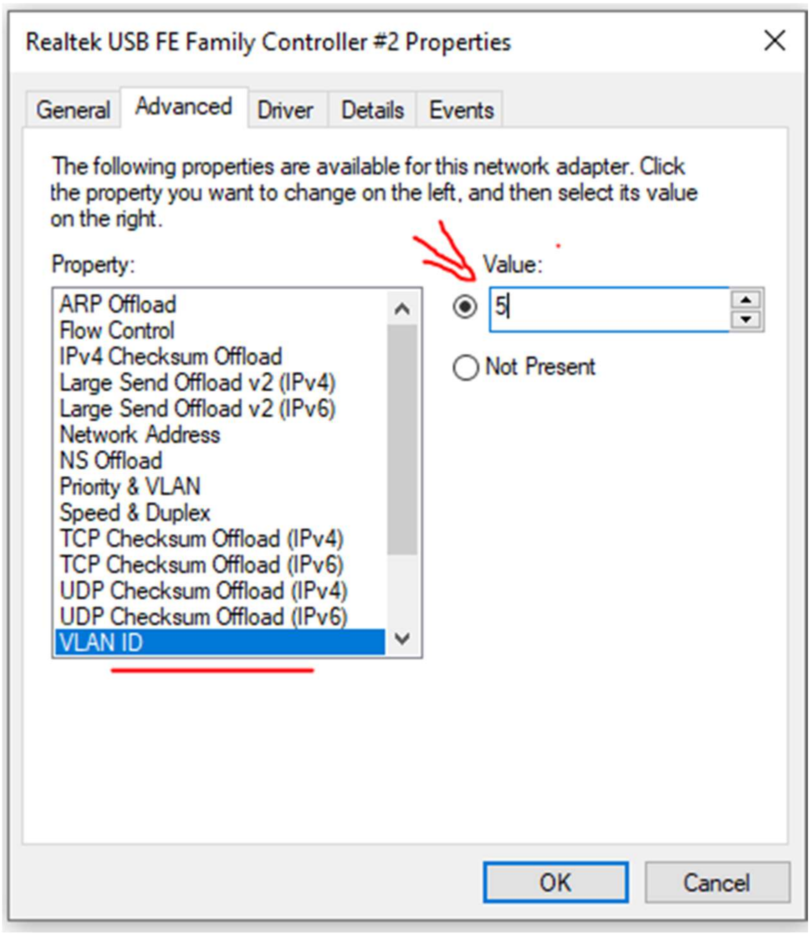
- [✓] Client for Microsoft Networks
- [✓] File and Printer Sharing for Microsoft Networks
- [✓] VirtualBox NDIS6 Bridged Networking Driver
- [✓] Npcap Packet Driver (NPCAP)
- [✓] QoS Packet Scheduler
- [✓] Bridge Driver
- [✓] Internet Protocol Version 4 (TCP/IPv4)

Install... | Uninstall | Properties

Description

Allows your computer to access resources on a Microsoft network.

OK | Cancel



Realtek USB FE Family Controller #2 Properties

General | Advanced | Driver | Details | Events

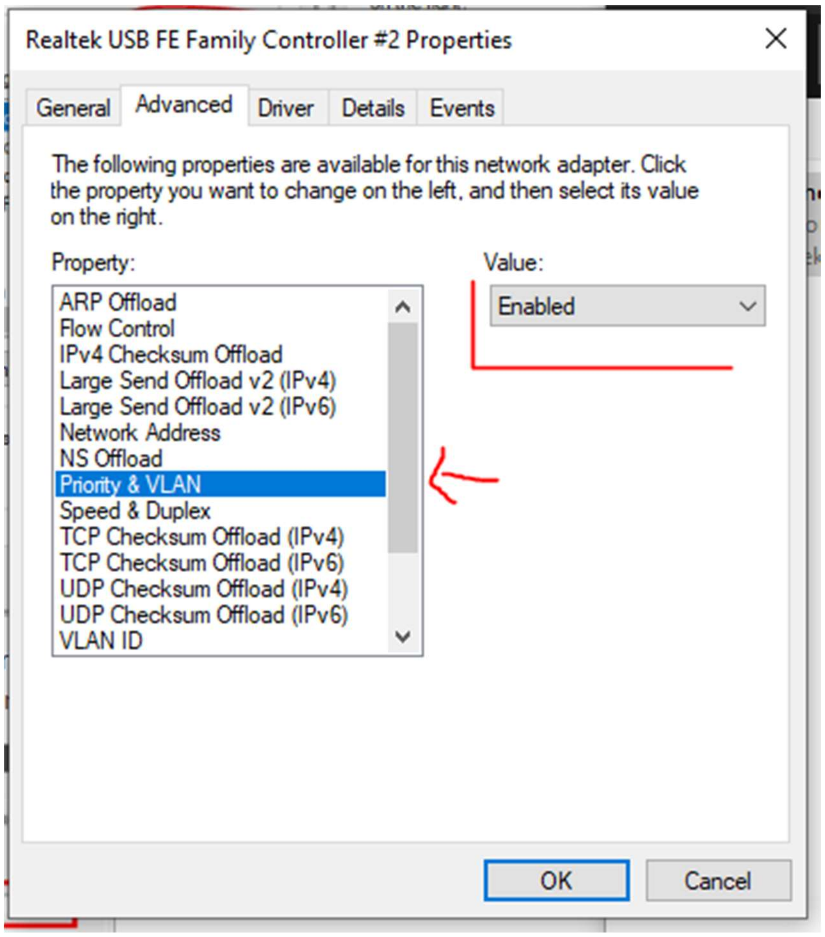
The following properties are available for this network adapter. Click the property you want to change on the left, and then select its value on the right.

Property:
- ARP Offload
- Flow Control
- IPv4 Checksum Offload
- Large Send Offload v2 (IPv4)
- Large Send Offload v2 (IPv6)
- Network Address
- NS Offload
- Priority & VLAN
- Speed & Duplex
- TCP Checksum Offload (IPv4)
- TCP Checksum Offload (IPv6)
- UDP Checksum Offload (IPv4)
- UDP Checksum Offload (IPv6)
- VLAN ID

Value: [✓ 5] Not Present

OK | Cancel

- Depending on the Ethernet device to be used for the VLAN test, the network device of the PC may require the VLAN function to be activated first,
- And in some cases, the network device driver must be updated to use the function.


Realtek USB FE Family Controller #2 Properties

The following properties are available for this network adapter. Click the property you want to change on the left, and then select its value on the right.

Property:
ARP Offload
Flow Control
IPv4 Checksum Offload
Large Send Offload v2 (IPv4)
Large Send Offload v2 (IPv6)
Network Address
NS Offload
Priority & VLAN
Speed & Duplex
TCP Checksum Offload (IPv4)
TCP Checksum Offload (IPv6)
UDP Checksum Offload (IPv4)
UDP Checksum Offload (IPv6)
VLAN ID

Value: Enabled

2. Source code

2.1 Python source code include 3 files:

- md_fw_declare.py
- md_fw_menu.py
- N16_1_10_1.py

parameter

This file will contain initial parameter of packets, include interface, MAC, IP, invalid IP... (please see source code)

md_fw_declare.py

```
from scapy.all import *
from scapy.layers.inet import *
from scapy.layers.inet6 import *
from random import randint
from netaddr import *
import binascii
import sys
import signal
from threading import Thread
from sqlalchemy import false

# Interface
IFACE                  = "Ethernet 2"

# Number of threads used
PKT_COUNT              = 5

# Scan Ports
FROM_PORT              = 1
TO_PORT                = 65536

# MAC Address
SRC_MAC                = "xx:xx:xx:xx:xx:xx"
DST_MAC                = "ff:ff:ff:ff:ff:ff" #Board
INVALID_SRC_MAC        = "fa:fb:fc:fd:fe:ff" #Invalid MAC

# VLAN ID
VLAN_ID                = 5

# IPv6s
INVALID_DST_IPv6       = "fd22:xxxx:xxx:3::xx" #Invalid IPv6
INVALID_SRC_IPv6       = "fd22:xxxx:xxx:3::xx" #Invalid IPv6

VALID_SRC_IPv6         = "fd22:xxxx:xxx:5::xx"
VALID_DST_IPv6         = "fd22:xxxx:xxx:5::xx"

VALID_DST_Multicast    = "ff02::1"
INVALID_DST_Multicast = "ff02::2"

# Ports
VALID_SPORT            = 13344
VALID_DPORT            = 13344
INVALID_DPORT          = 13456
INVALID_SPORT          = 13456

RANGE                  = (1000, 65535)

pro_type               = TCP

# Layers
dot1q                  = Dot1Q(vlan=VLAN_ID)

# Payload
payload_default        ="Default"

PKT_Default_Receive = Ether()/dot1q/IPv6(src=VALID_SRC_IPv6,
dst=VALID_DST_IPv6)/pro_type(sport=VALID_SPORT, dport=VALID_DPORT)/payload_default
PKT_Default_Send = Ether(dst=SRC_MAC,src=DST_MAC)/dot1q/IPv6(src=VALID_DST_IPv6,
dst=VALID_SRC_IPv6)/pro_type(sport=VALID_DPORT, dport=VALID_SPORT)/payload_default
```

md_fw_menu.py

This file will contain a menu option, It just have common menu with packet infor and send packet, if your SyRS require more than that, please refer to it and make another one in main file.

menu

```
def print_menu():
    cloop=True
    while cloop:
        print (22 * "-" , "MENU" , 22 * "-")
        print ("\t1. [Infor]     {:<24} ".format("Packet information"))
        print ("\t2. [Send]      {:<24} ".format("Packet Send"))
        print ("\t0. [Exit]      {:<24} ".format("Exit"))

        print (50 * "-")
        try:
            choice =input("Enter your choice [0-2]: ")
            if (int(choice) >=0 and int(choice) <=2):
                cloop = False
        except ValueError:
            print('')
    return choice
if __name__=='__main__':
    print_menu()
```

N16_1_15_2.py

This is main file of script, It uses scapy to create packet and send via ethernet

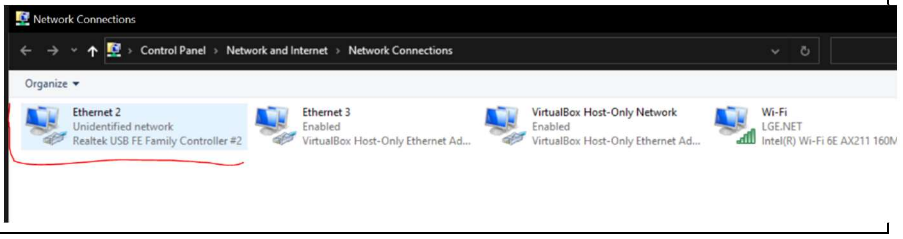
main

```
from md_fw_declare import *
from md_fw_menu import *

#VALID UDP PORT
# PKT_Default_Receive=Ether()/dot1q/IPv6(src=VALID_SRC_IPv6,
dst=VALID_DST_IPv6)/UDP(sport=VALID_SPORT,dport=VALID_DPORT)/payload_default

#INVALID UDP PORT
PKT_Default_Receive=Ether()/dot1q/IPv6(src=VALID_SRC_IPv6,
dst=VALID_DST_IPv6)/UDP(sport=VALID_SPORT,dport=12345)/payload_default
def print_infor():
    try:
        global PKT_Default_Receive
        print("\n----------Packet-information-------------")
        PKT_Default_Receive.show()
    except Exception as ex:
        print("Error:"+ex)
#16.1.15.2 Undefined UDP port message handling
def send_packet():
    global PKT_Default_Receive
    try:
        PKT_Default_Receive.show()
        sendp(PKT_Default_Receive, iface=IFACE)
    except:
        print("Error: Please Connect Ethernet...")
def main():
    cloop=True
    while cloop:
        try:
            choice = print_menu()
            if int(choice)      ==1:
                print_infor()
            elif int(choice)    ==2:
                send_packet()
            elif int(choice)    ==0:
                cloop=False
        except KeyboardInterrupt:
            print ('\nThanks! See you later!\n\n')
            cloop=False

if __name__ == '__main__':
    main()
```

2.2 Set up parameter

In md_fw_declare.py file, we need to set up parameter in order for script work properly

field	comment

# Interface 	
IFACE = "Ethernet 2"
	modify IFACE value to match with your PC's ethernet interface
	 Network Connections
	Control Panel › Network and Internet › Network Connections
	Organize
	Ethernet 2
	Unidentified network
	Realtek USB FE Family Controller #2
	Ethernet 3
	Enabled
	VirtualBox Host-Only Ethernet Ad...
	VirtualBox Host-Only Network
	Enabled
	VirtualBox Host-Only Ethernet Ad...
	Wi-Fi
	LGE.NET
	Intel(R) Wi-Fi 6E AX211 160M

# MAC Address 	
SRC_MAC = "xx:xx:xx:xx:xx:xx"
DST_MAC = "ff:ff:ff:ff:ff:ff" #Board
	• If packet doesn't specify MAC address, they will be generated automatically
	
	PKT_Default_Receive = Ether()/dot1q/IPv6(src=VALID_SRC_IPv6, dst=VALID_DST_IPv6)/pro_type(sport=VALID_SPORT, dport=VALID_DPORT)/payload_default
	
	• If packet specify MAC address, they will be used directly
	
	PKT_Default_Receive = Ether(dst=SRC_MAC,src=DST_MAC)/dot1q/IPv6(src=VALID_SRC_IPv6, dst=VALID_DST_IPv6)/pro_type(sport=VALID_SPORT, dport=VALID_DPORT)/payload_default

# VLAN ID 	
VLAN_ID = 5
	modify it as VLAN id that you want to test

# IPv6s 	
INVALID_SRC_IPv6 = "fd22:xxxx:xxx:3::11" #Invalid IPv6
VALID_SRC_IPv6 = "fd22:xxxx:xxx:5::10"
VALID_DST_IPv6 = "fd22:xxxx:xxx:5::14"
VALID_DST_Multicast = "ff02::1"
INVALID_DST_Multicast = "ff02::2"
	IPv6 address

# Ports 	
VALID_SPORT = 13344
VALID_DPORT = 13344
	source port and dest port of packet
	
	can add invalid port

# Payload 	
payload_default ="Default"
	change payload content or leave it blank for testing empty payload packet

# Layers 	
dot1q = Dot1Q(vlan=VLAN_ID)
	This is Vlan of packet.
	
	If packet have VLan ID and PC has turned on VLAN ID too, it will send double VLAN ID
	
	Just use one VLAN ID, you should remove VLAN ID in PC setup

3. Demo

3.1 Test receiving packet

test step

N16_1_6_7.py

```
from md_fw_declare import *
from md_fw_menu import *
PKT_Default_Receive[IPv6].src=INVALID_SRC_IPv6
def print_infor():
    try:
        global PKT_Default_Receive
        print("\n----------Packet-information-------------")
        PKT_Default_Receive.show()
    except Exception as ex:
        print("Error:"+ex)
#16.1.13.4 Undefined target IPv6 address
def send_packet():
    global PKT_Default_Receive
    try:
        PKT_Default_Receive.show()
        sendp(PKT_Default_Receive, iface=IFACE)
    except:
        print("Error: Please Connect Ethernet...")
def main():
    cloop=True
    while cloop:
        try:
            choice = print_menu()
            if int(choice)      ==1:
                print_infor()
            elif int(choice)    ==2:
                send_packet()
            elif int(choice)    ==0:
                cloop=False
        except KeyboardInterrupt:
            print ('\nThanks! See you later!\n\n')
            cloop=False

if __name__ == '__main__':
    main()

#Check Log to show result
#   tail /data/log_data/ulogd/full.log
# Sep 20 20:04:36 Input DROP IN=eth0.5 OUT=
MAC=ff:ff:ff:ff:ff:ff:xx:xx:xx:xx:xx:xx:xx:xx:xx:00:00:00 SRC=fd22:xxxx:xxx:3::10
DST=fd22:xxxx:xx:5::14 LEN=67 TC=0 HOPLIMIT=64 FLOWLBL=0 PROTO=TCP SPT=13344
DPT=13344 SEQ=0 ACK=0 WINDOW=8192 SYN URGP=0 MARK=0
```

test step

```
[Pre-conditions]
```

```
1. ECU Power On and connect to Radmoon 2.
2. KL 15 : ON
3. Setup static IPv6 on PC
4. Connect to HU via console (skip all Apenddix steps)
5. Download and extract python script for testing

***TEST RECEIVING PACKET****
[Test Steps]

1. Read current packet record.
$ ip6tables -nvL INPUT
$ ip6tables -nvL INVALID_IPV6
2. Run script to send an Invalid data definition packet.
a. open command prompt at python script folder
b. run command:
python N16_1_6_7.py
c. input 2 to send an Invalid TCP flags packet

[Expected Results]
Verify that Packets with not specified IPv6 TCP in the data definition to be
discarded
1. Read packet record after sending 1 packet.
$ ip6tables -nvL INPUT
$ ip6tables -nvL INVALID_IPV6
You can see INVALID_IPV6 has increase by 1
2. View log to show detail:
$ tail /log/log_data/ulogd/full.log
```

3.2 Test sending packet

source code

N44_1_1_1.py

```
#44.1.1.1
#When sending UDP messages, the destination address must be corrected
#check UDP source port
#sending

import socket, time, os, struct
from ipaddress import ip_address
try:
    from itertools import izip_longest as zip_longest
except ImportError:
    from itertools import zip_longest

# IPv6s
INVALID_DST_IPv6      = "fd22:xxxx:xxx:3::15" #Invalid IPv6
INVALID_SRC_IPv6      = "fd22:xxxx:xxx:3::12" #Invalid IPv6
VALID_SRC_IPv6        = "fd22:xxxx:xxx:5::14"
VALID_DST_IPv6        = "fd22:xxxx:xxx:5::10"
```

```
VALID_SPORT = 13344
VALID_DPORT = 13344

INVALID_SPORT = 13455
INVALID_DPORT = 13455

MULTICAST_TTL = 32

VALID_MULTICAST = "ff02::1"
INVALID_MULTICAST = "ff02::2"


mes = "Default"

# Next headers for IPv6 protocols
IPV6_NEXT_HEADER_HOP_BY_HOP = 0
IPV6_NEXT_HEADER_TCP = 6
IPV6_NEXT_HEADER_UDP = 17
IPV6_NEXT_HEADER_ICMP = 58

UPPER_LAYER_PROTOCOLS = [
    IPV6_NEXT_HEADER_TCP,
    IPV6_NEXT_HEADER_UDP,
    IPV6_NEXT_HEADER_ICMP,
]

# ICMP Protocol codes
ICMP_ECHO_REQUEST = 128
ICMP_ECHO_RESPONSE = 129

# Default hop limit for IPv6
HOP_LIMIT_DEFAULT = 64


def checksum_calculate(data):
    # Create halfwords from data bytes. Example: data[0] = 0x01, data[1] = 0xb2 =>
0x01b2
    halfwords = [
        ((byte0 << 8) | byte1)
        for byte0, byte1 in zip_longest(data[::2], data[1::2], fillvalue=0x00)
    ]

    checksum = 0
    for halfword in halfwords:
        checksum += halfword
        checksum = (checksum & 0xFFFF) + (checksum >> 16)

    checksum ^= 0xFFFF

    if checksum == 0:
        return 0xFFFF
    else:
        return checksum


class IPv6Header():
    _version = 6
    _header_length = 40

    def __init__(self,
                 source_address,
                 destination_address,
                 traffic_class=0,
                 flow_label=0,
                 hop_limit=64,
                 payload_length=0,
                 next_header=0):
        self.version = self._version
        self._source_address = self._convert_to_ipaddress(source_address)
        self._destination_address = self._convert_to_ipaddress(
            destination_address)
        self.traffic_class = traffic_class
        self.flow_label = flow_label
        self.hop_limit = hop_limit
        self.payload_length = payload_length
        self.next_header = next_header

    def _convert_to_ipaddress(self, value):
        if isinstance(value, bytearray):
            value = bytes(value)

        return ip_address(value)

    @property
    def source_address(self):
        return self._source_address

    @source_address.setter
    def source_address(self, value):
        self._source_address = self._convert_to_ipaddress(value)

    @property
    def destination_address(self):
        return self._destination_address

    def pack(self):
        data = bytearray([
            ((self.version & 0x0F) << 4) | ((self.traffic_class >> 4) & 0x0F),
            ((self.traffic_class & 0x0F) << 4) |
            ((self.flow_label >> 16) & 0x0F), ((self.flow_label >> 8) & 0xFF),
            ((self.flow_label & 0xFF))
        ])
        data += struct.pack(">H", self.payload_length)
        data += bytearray([self.next_header, self.hop_limit])
        data += self.source_address.packed
        data += self.destination_address.packed

        return data

    @classmethod
    def unpack(cls, data):
        b = bytearray(data.read(4))
        version = (b[0] >> 4) & 0x0F
        traffic_class = ((b[0] & 0x0F) << 4) | ((b[1] >> 4) & 0x0F)
        flow_label = ((b[1] & 0x0F) << 16) | (b[2] << 8) | b[3]

        payload_length = struct.unpack(">H", data.read(2))[0]
        next_header = ord(data.read(1))
        hop_limit = ord(data.read(1))
        src_addr = bytearray(data.read(16))
        dst_addr = bytearray(data.read(16))

        return cls(src_addr, dst_addr, traffic_class, flow_label, hop_limit,
                   payload_length, next_header)

    def __repr__(self):
        return "IPv6Header(source_address={}, destination_address={}, 
next_header={}, payload_length={}, \
            hop_limit={}, traffic_class={}, flow_label={})".format(
            self.source_address.compressed, self.destination_address.compressed,
            self.next_header, self.payload_length, self.hop_limit,
            self.traffic_class, self.flow_label)

    def __len__(self):
        return self._header_length
    def display_info(self):
        print("###[ IPv6 ]###")
        print("version    = " + str(self.version))
        print("tc         = " + str(self.traffic_class))
        print("fl         = " + str(self.flow_label))
        print("plen       = " + str(self.payload_length))
        print("nh         = " + str(self.next_header))
        print("hlim       = " + str(self.hop_limit))
        print("src        = " + str(self._source_address))
        print("dst        = " + str(self._destination_address))

class TCPHeader():

    @property
    def type(self):
        return 6

    def __init__(self, src_port, dst_port, payload, seq=0, ack_seq=0, doff=0x5, 
fin=0, syn=1, rst=0, psh=0, ack=0, urg=0, window=8192, checksum=0, urg_ptr=0):
        self.src_port = src_port
        self.dst_port = dst_port
        self.seq = seq
        self.ack_seq = ack_seq
        self.doff = doff
        self.fin = fin
        self.syn = syn
        self.rst = rst
        self.psh = psh
        self.ack = ack
        self.urg = urg
        self.window = window
        self.checksum = checksum
        self.urg_ptr = urg_ptr
        self.payload = payload
        self.length = 20+len(self.payload)

        self.offset_res = (self.doff << 4) + 0
        self.flags = self.fin + (self.syn << 1) + (self.rst << 2) + \
            (self.psh << 3) + (self.ack << 4) + (self.urg << 5)

        # the ! in the pack format string means network order
        self.header = struct.pack('!HHLLBBHHH' , self.src_port, self.dst_port, 
self.seq, self.ack_seq, self.offset_res, self.flags,  self.window, self.checksum, 
self.urg_ptr)

    def pack(self, source, destination):
        data_offset = self.offset_res
        flags = self.fin + (self.syn << 1) + (self.rst << 2) + \
            (self.psh << 3) + (self.ack << 4) + (self.urg << 5)
        tcp_header = struct.pack('!HHLLBBHHH',
                                 self.src_port,
                                 self.dst_port,
                                 self.seq,
                                 self.ack,
                                 data_offset,
                                 flags,
                                 self.window,
                                 self.checksum,
                                 self.urg)
        # pseudo header fields
        source_ip = source
        destination_ip = destination
        reserved = 0
        protocol = socket.IPPROTO_TCP
        total_length = len(tcp_header) + len(self.payload)

        # Pseudo header
        psh = ip_address(source_ip).packed
        psh += ip_address(destination_ip).packed
        psh += struct.pack(">I", reserved)
        psh += struct.pack(">I", total_length)
        psh += struct.pack(">I", protocol)

        psh = psh + tcp_header + self.payload.encode('ascii')
        tcp_checksum = checksum_calculate(psh)
        tcp_checksum = (tcp_checksum >> 8) +((tcp_checksum << 8)&0xFFFF)
        tcp_header = struct.pack("!HHLLBBH",
                                 self.src_port,
                                 self.dst_port,
                                 self.seq,
                                 self.ack,
                                 data_offset,
                                 flags,
                                 self.window)
        tcp_header += struct.pack('H', tcp_checksum) + \
            struct.pack('!H', self.urg) + self.payload.encode('ascii')
        return tcp_header
    def display_info(self):
        print("###[ TCP ]###")
        print("sport     = " + str(self.src_port))
        print("dport     = " + str(self.dst_port))
        print("seq       = " + str(self.seq))
        print("ack       = " + str(self.ack))
        print("dataofs   = " + str(self.offset_res))
        print("reserved  = " + str(0))
        print("flags     = " + str(self.flags))
        print("window    = " + str(self.window))
        print("chksum    = " + str(self.checksum))
        print("urgptr    = " + str(self.urg))
        print("options   = ''")
        print("###[ Raw ]###")
        print(self.payload)

class UDPHeader(object):
    def __init__(self, src_port, dst_port, payload):
        self.src_port = src_port
        self.dst_port = dst_port
        self.payload = payload
        self.checksum = 0
        self.length = 8  # UDP Header length

    def pack(self, src_port, dst_port, proto=socket.IPPROTO_UDP):
        length = 8 + len(self.payload)
        packet = struct.pack('!HHHH',
                             self.src_port, self.dst_port, length, self.checksum)
        packet += self.payload.encode('ascii')
        return packet
    def check_sum(self, src_addr, dst_addr):
        # Pseudo header
        length = len(self.payload) + 8
        psh = ip_address(src_addr).packed
        psh += ip_address(dst_addr).packed
        psh += struct.pack(">H", socket.IPPROTO_UDP)
        psh += struct.pack(">I", length)

        #UDP header
        psh += struct.pack(">H", self.src_port)
        psh += struct.pack(">H", self.dst_port)
        psh += struct.pack(">I", length)
        psh += struct.pack(">I", 0x00)
        psh += self.payload.encode('ascii')
        self.checksum = checksum_calculate(psh)
        return checksum_calculate(psh)

    def display_info(self):
        print("###[ UDP ]###")
        print("sport         = " + str(self.src_port))
        print("dport         = " + str(self.dst_port))
        print("len           = " + str(self.length))
        print("chksum        = " + str(hex(self.checksum)))

def send_udp_packet(src_addr, dst_addr, src_port, dst_port, data="Default"):
    # create UPD frame
    try:
        ipv6_hdr = IPv6Header(src_addr, dst_addr, next_header=socket.IPPROTO_UDP,
payload_length=8 + len(data))
        udp_hdr = UDPHeader(src_port, dst_port, payload=data)
        udp_hdr.check_sum(src_addr, dst_addr)
        frame=ipv6_hdr.pack() + udp_hdr.pack(src_port, dst_port)
        ipv6_hdr.display_info()
        udp_hdr.display_info()
    except Exception as exx:
        print(exx)
    #send UDP packet
    try:
        sock = socket.socket(socket.AF_INET6, socket.SOCK_RAW, 255)
        sock.sendto(frame, (dst_addr, 0))
        sock.close()
        time.sleep(2)
        print("\n...Sent 1 Packet")
    except Exception as exx:
        time.sleep(2)
        print(exx)

def send_tcp_packet(src_addr, dst_addr, src_port, dst_port, data="Default"):
    try:
        # create TCP frame
        ipv6_hdr = IPv6Header(src_addr, dst_addr, next_header=socket.IPPROTO_TCP,
payload_length=20 + len(data))
        tcp_hdr = TCPHeader(src_port, dst_port, payload=data)
        frame=ipv6_hdr.pack() + tcp_hdr.pack(src_addr, dst_addr)
        ipv6_hdr.display_info()
        tcp_hdr.display_info()
    except Exception as exx:
        print(exx)
    try:
        #send tcp packet
        sock = socket.socket(socket.AF_INET6, socket.SOCK_RAW, 255)
        sock.sendto(frame, (dst_addr, 0))
        sock.close()
        time.sleep(2)
        print("\n...Sent 1 Packet")
    except Exception as exx:
        time.sleep(2)
        print(exx)


def print_menu():
    cloop = True
    while cloop:
        print(22 * "-", "MENU", 22 * "-")
        print("\t1. [Send]      {:<24} ".format("Send A Valid UDP Packet"))
        print("\t2. [Send]      {:<24} ".format("Send An Invalid destination IPv6
UDP Packet"))
        print("\t3. [Send]      {:<24} ".format("View log
(/log/log_data/ulogd/full.log)"))
        print("\t4. [Send]      {:<24} ".format("View valid packet counter "))
        print("\t5. [Send]      {:<24} ".format("View invalid packet counter "))
        print("\t0. [Exit]      {:<24} ".format("Exit"))
        print(50 * "-")
        try:
            choice = input("Enter your choice [0-5]: ")
            if (int(choice) >= 0 and int(choice) <= 5):
                cloop = False
        except ValueError:
            print('')
    return choice

def main():
    cloop = True
    while cloop:
        try:
            choice = print_menu()
            if int(choice) == 1:

send_udp_packet(src_addr=VALID_SRC_IPv6,dst_addr=VALID_DST_IPv6,src_port=VALID_SPO
RT,dst_port=VALID_DPORT)
            elif int(choice) == 2:
                # UDP invalid des IP

send_udp_packet(src_addr=VALID_SRC_IPv6,dst_addr=INVALID_DST_IPv6,src_port=VALID_S
PORT,dst_port=VALID_DPORT)
            elif int(choice) == 3:
                try:
                    os.system('tail /log/log_data/ulogd/full.log')
                except Exception as exx:
                    print(exx)
            elif int(choice) == 4:
                try:
                    os.system('ip6tables -nvL')
                except Exception as exx:
                    print(exx)
            elif int(choice) == 5:
                try:
                    os.system('ip6tables -nvL INVALID_IPV6')
                except Exception as exx:
                    print(exx)
            elif int(choice) == 0:
                cloop = False

        except KeyboardInterrupt:
            print('\nThanks! See you later!\n\n')
            cloop = False


if __name__ == '__main__':
    main()
```

test step

```
[Pre-conditions]
1. Connect the packet generator (ex. RADMOON 2) to HU.
2. Setup static IPv6 on PC
3. Connect to HU via console (skip all Apenddix steps)
4. Download and extract python script for testing

[Test Steps]
1. create script file inside device:
1.a. In console run below commands
mount -o rw,remount /
```

```
cat>N44_1_1_1_S.py
1.b. copy text in N44_1_1_1_S.py file and paste (right mouse click to paste)
1.c. ctrl+D to complete.

2. run command:
python3 N44_1_1_1_S.py
input 3 to view log before sending
input 4 to see valid multicast packet counter
input 1 to send a valid multicast UDP packet
input 4 to see whitelist packet counter after sending a packet, counter in ACCEPT
UDP will increase by 1

input 5 to see invalid UDP packet counter
input 2 to send an invalid source_port packet
input 5 to see whitelist packet counter after sending a packet, counter in
INVALID_IPV6 will increase by 1

input 3 to see new log appear after sending an invalid multicast packet
[Expected Results]
expected results has described above
```

B. Questions, Exercises

TBD

C. References

No.	Info	Link/ file/ name of ebook
	tcpdump	https://opensource.com/article/18/10/introduction-tcpdump
	install PIP	https://www.geeksforgeeks.org/how-to-install-pip-on-windows/
	tcpdump man	https://www.tcpdump.org/manpages/tcpdump.1.html