

False Data Injection Attack

Two cases: Simulation and Emulation

Two Types: Payload modification and Header modification

Software: NetSim Standard v13.3 (64 bit), Visual Studio 2022

Project code download link: https://github.com/NetSim-TETCOS/False-Data-Injection-Attack-in-Internetworks_v13.3/archive/refs/heads/main.zip

Follow the instructions specified in the following link to download and setup the Project in NetSim:

<https://support.tetcos.com/en/support/solutions/articles/14000128666-downloading-and-setting-up-netsim-file-exchange-projects>

Introduction

FDI (False Data Injection) attack is a type of cyber-attack where an attacker injects false data into a system or network with the intent of causing damage or disruption. FDI attacks can be launched against various types of systems, including industrial control systems, critical infrastructure, financial systems, and information systems.

In an FDI attack, the attacker may modify or manipulate data in transit or at rest to achieve their objectives. For example, an attacker may alter the data in a financial transaction to redirect funds to a different account, modify the configuration of an industrial control system to cause physical damage, or manipulate data in a way that causes a system to crash or malfunction.

Toy Example: FDI Attack on PING

In this example, we launch an FDI attack on ICMP ping messages between a source and destination. The destination receives the message and processes it as if it were legitimate.

Case 1: Virtual implementation within NetSim simulator using by capturing real PING traffic using Wireshark.

This case is a simpler method of simulating the FDI attack requiring only one machine. Case 2 (described later) involves using 3 machines.

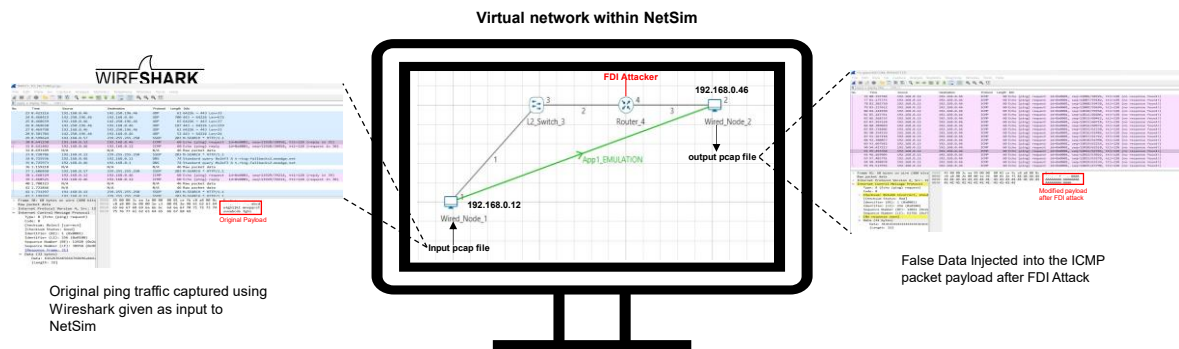


Fig 1: PING application between a real source and real destination is captured as a pcap file and given as an input to a virtual source inside NetSim. In this example, the source IP is set to 192.168.0.12 and the destination IP is set to 192.168.0.46. The external pcap file is available in the project download link. **Fig needs changes**

Generating Packet capture for NetSim

We explain the steps used to capture PING data as a pcap file. This has been provided for those readers who may wish to capture their own pcap files and use implement the FDI attack on that.

1. Open Wireshark in the system where NetSim is installed.
2. Once the Wireshark is opened, please select the proper interface. (For Ex: Ethernet) as show below. Double click on the interface to open live packet capture window.

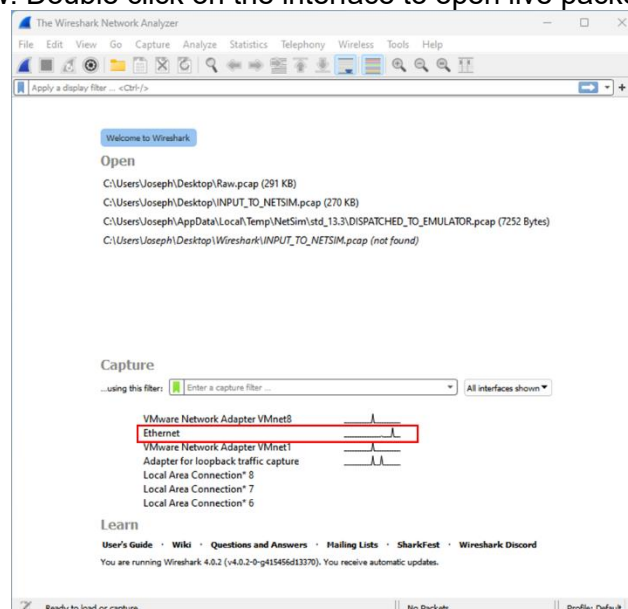


Fig 2: Select packet capture interface to capture packets at source.

3. In this Example we have considered areal source with 192.168.0.12 and a real destination with IP 192.168.0.46. Open command line at source device and enter the command
 ➤ ping 192.168.0.46 -t

```

Command Prompt - ping 192.168.0.46 -t
Microsoft Windows [Version 10.0.19045.2604]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ADMIN>ping 192.168.0.46 -t

Pinging 192.168.0.46 with 32 bytes of data:
Reply from 192.168.0.46: bytes=32 time<1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=5ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=5ms TTL=128
Reply from 192.168.0.46: bytes=32 time=2ms TTL=128
Reply from 192.168.0.46: bytes=32 time<1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time<1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time=1ms TTL=128
Reply from 192.168.0.46: bytes=32 time<1ms TTL=128

```

Fig 3 : Ping traffic between source IP 192.168.0.12 and destination IP 192.168.0.46

4. The pcap file will contain all incoming and outgoing packets from the system in which the capture is being done. Once the ping traffic capture is finished stop the Wireshark packet capture and save the packet capture in a desired location with desired name (*.pcap) for E.g., Raw.pcap with Save as type as **Wireshark/tcpdump.... -pcap**.
5. This PCAP needs to be edited before giving as input to NetSim. The editcap application in Wireshark Installation Directory can be used to edit the any pcap file to be provided as a input to NetSim
6. Go to Wireshark installation directory [C:\Program Files\Wireshark]
7. Open command prompt, and execute the following command:
 - **editcap -C 14 -L -T rawip -F pcap "<File Location where the file is present>/Raw.pcap" "<File Location where the file needs to be saved>/INPUT_TO_NETSIM.pcap"**

Steps to simulate

1. Go to start search Run → Enter the command "SystemPropertiesAdvanced" and then click on OK.
2. Click the Environment Variables → Add the following Environment PATH variable.
 <File-Path-where-INPUT_TO_NETSIM.pcap file is located>\INPUT_TO_NETSIM.pcap
 For eg: C:\Users\Joseph\Desktop\INPUT_TO_NETSIM.pcap

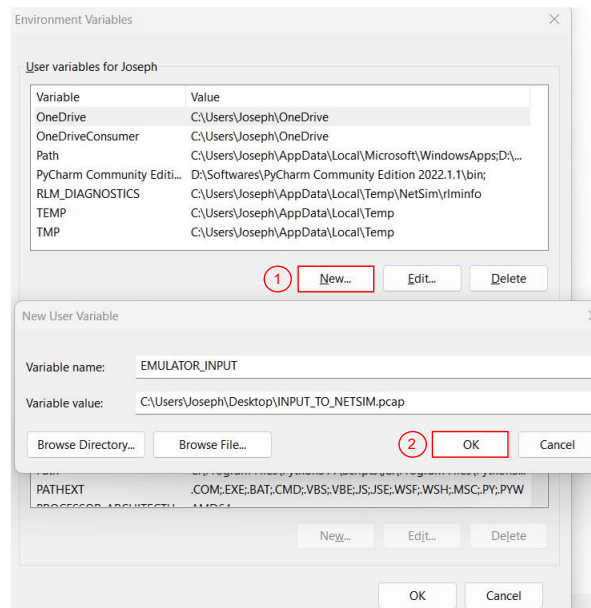


Fig 4 : Environment Variable Path

For more information how to provide pcap file as input refer our knowledge base article <https://support.tetcos.com/support/solutions/articles/14000103748-how-can-i-provide-pcap-file-as-input-to-simulation->

Packet capture playback in NetSim

- The **FDI_Attack_in_Internetworks_v13.3** comes with a sample network configuration that are already saved. To open this example, go to Your work in the home screen of NetSim and click on the **FDI_Sample_Internetwork** from the list of experiments.
- The saved network scenario consists of
 - a. 2 Wired Node
 - b. 1 L2 Switch
 - c. 1 Router

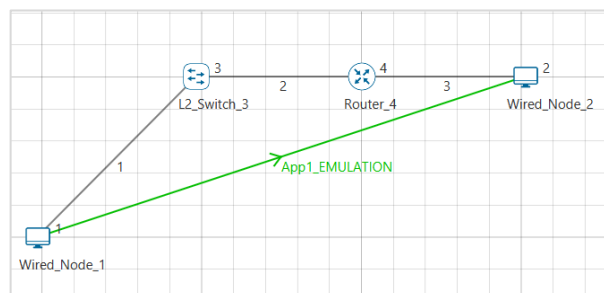


Fig 5: NetSim Emulation Scenario, Wired_Node_1 device mapped for Source IP 192.168.0.12 and Wired_Node_2 device mapped for Destination IP 192.168.0.46

1. Application Properties
 - Application Type - EMUALTION
 - Source IP - 192.168.0.12
 - Destination IP – 192.168.0.46
2. Run the Simulation for 100 sec.

Observations

After the simulation is completed, you can observe the results using Wireshark captured files. In the Result Dashboard, On the left side, Packet Capture → Emulation and you can see all Emulated Packets captured.

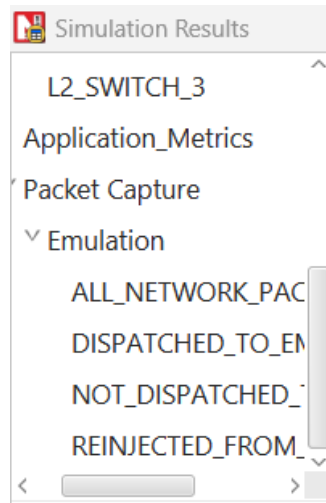


Fig 6: Emulation Packet Capture in Result Dashboard

You can observe original packets in the DISPATCHED_TO_EMUALTOR.pcap file.

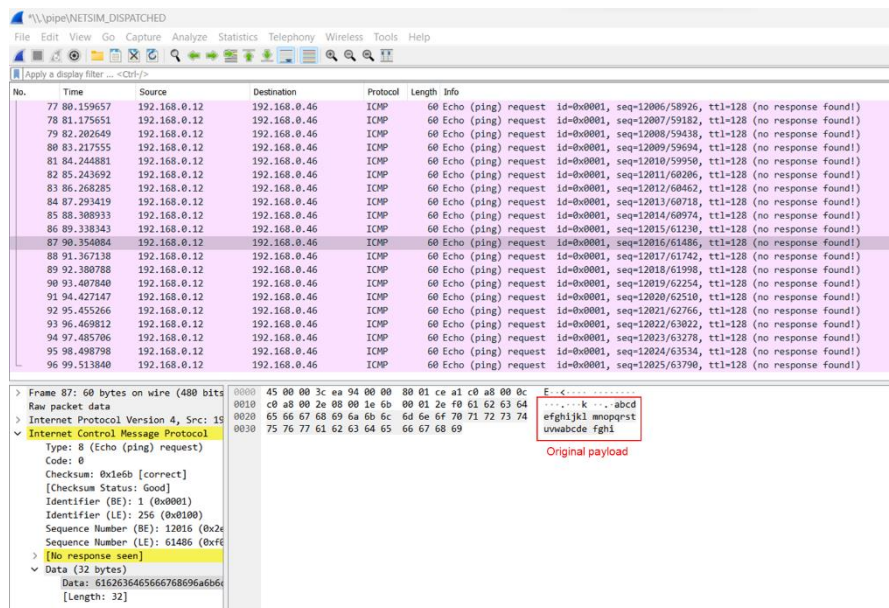


Fig 7: Original payload captured by NetSim emulator

You can observe false data injected packets in the REINJECTED_FROM_EMUALTOR.pcap file

No.	Time	Source	Destination	Protocol	Length	Info
76	80.559784	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12006/58926, ttl=128 (no response found!)
77	81.175735	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12007/59182, ttl=128 (no response found!)
78	82.202710	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12008/59438, ttl=128 (no response found!)
79	83.217612	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12009/59694, ttl=128 (no response found!)
80	84.244950	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12010/59950, ttl=128 (no response found!)
81	85.243761	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12011/60206, ttl=128 (no response found!)
82	86.268335	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12012/60462, ttl=128 (no response found!)
83	87.293518	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12013/60718, ttl=128 (no response found!)
84	88.309069	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12014/60974, ttl=128 (no response found!)
85	89.338406	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12015/61230, ttl=128 (no response found!)
86	90.354159	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12016/61486, ttl=128 (no response found!)
87	91.367218	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12017/61742, ttl=128 (no response found!)
88	92.380857	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12018/61998, ttl=128 (no response found!)
89	93.407942	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12019/62254, ttl=128 (no response found!)
90	94.427213	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12020/62510, ttl=128 (no response found!)
91	95.455394	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12021/62766, ttl=128 (no response found!)
92	96.469905	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12022/63022, ttl=128 (no response found!)
93	97.485791	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12023/63278, ttl=128 (no response found!)
94	98.498878	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12024/63534, ttl=128 (no response found!)
95	99.513956	192.168.0.12	192.168.0.46	ICMP	60	Echo (ping) request id=0x0001, seq=12025/63790, ttl=128 (no response found!)

Frame 91: 60 bytes on wire (480 bits)	Raw packet data	Internet Protocol Version 4, Src: 192.168.0.12, Dst: 192.168.0.68	Internet Control Message Protocol
0000	45 00 00 3c na 99 00 00 00 01 ce 9c c0 a8 00 0c	0000	45 00 00 3c na 99 00 00 00 01 ce 9c c0 a8 00 0c
0010	c0 a8 00 2e 08 00 1e 66 00 01 2e f5 41 41 41 41	0010	c0 a8 00 2e 08 00 1e 66 00 01 2e f5 41 41 41 41
0020	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	0020	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41
0030	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	0030	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41

Modified payload after FDI attack

Fig 8: Traffic with false data injected. Observe the difference in payload.

Case 2: FDI implementation in NetSim emulator. We have 3 systems – source, destination, and Emulator. The PING packets from source to destination pass through the emulator.

FDI attack on real traffic using NetSim Emulator

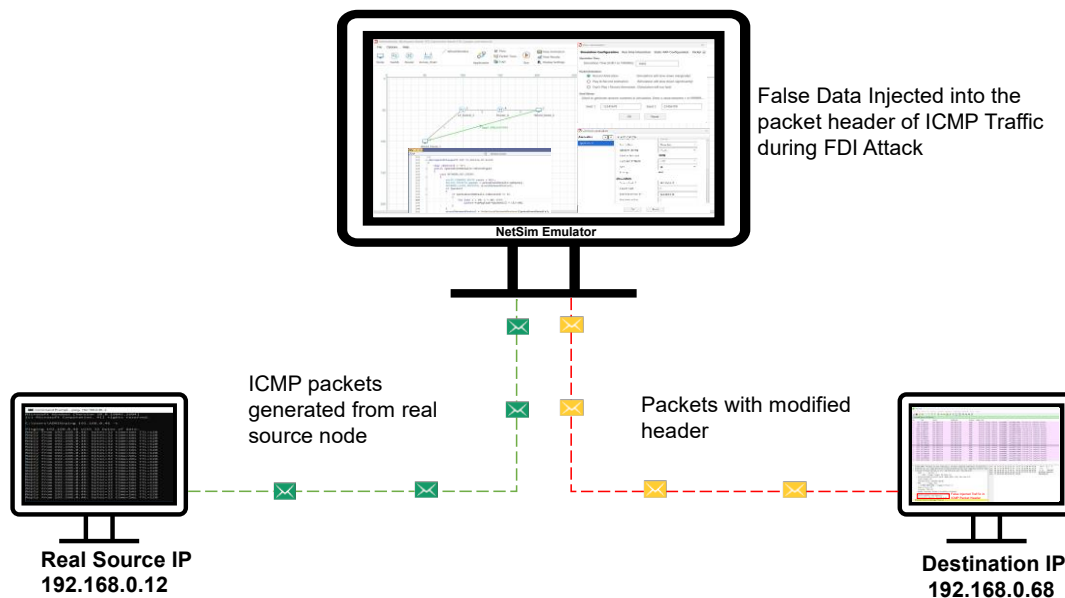


Fig 9 : PING application between source and destination. The source IP is set to 192.168.0.12 and the destination IP is set to 192.168.0.46. In NetSim We are implementing FDI Attack by modifying the destination IP address to 192.168.0.68 in ICMP packet header.

Steps to Simulate

The set-up to run emulation would be to have a minimum of three (3) PC's. One would be the real source, the second would run NetSim emulation server, and the third would be the real destination.

In this Example, we have considered 3 systems as shown below.

Real Source IP : 192.168.0.12

NetSim Emulation Server IP : 192.168.0.81

Real Destination IP : 192.168.0.46

Setting up the NetSim Emulation Server

1. Run the NetSim in Administrator Mode (Right Click on NetSim Icon → Run as Administrator)
2. Open the Existing Sample **FDI_Sample_Internetwork** from the list of Experiments (In NetSim Home Screen → Your Work)
3. The saved network scenario consists of
 - d. 2 Wired Node
 - e. 1 L2 Switch
 - f. 1 Router

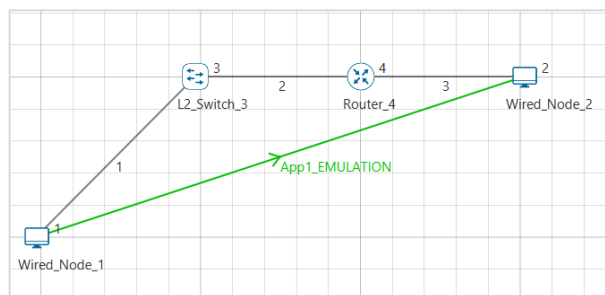


Fig 10: NetSim Emulation Scenario, Wired_Node_1 device mapped for Source IP 192.168.0.12 and Wired_Node_2 device mapped for Destination IP 192.168.0.46

4. Application Properties
 - Application Type - EMUALTION
 - Source IP - 192.168.0.12
 - Destination IP – 192.168.0.46
5. Run the Simulation for 100 sec.

Setting up the Real Source and Destination

The client systems which are sources of real traffic can be connected to NetSim emulator by resetting the gateway. Once the gateway for the client system is set as the NetSim Emulator PC then traffic from the clients will go via NetSim Emulator PC.

Configuring NetSim Emulator as a Gateway in NetSim in Windows clients

1. Open command prompt in Administrator Mode
2. Type the command
 - a. **route add 192.168.0.46 mask 255.255.255.0 192.168.0.81 metric 1**
 - b. After the Execution , you will get “OK”.

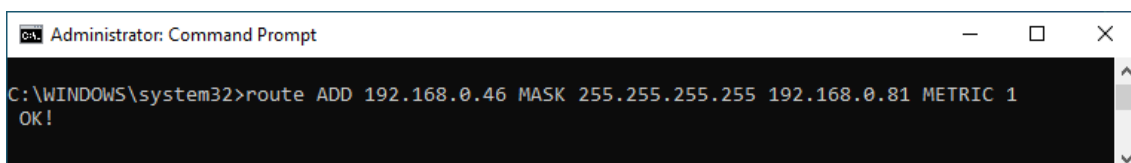
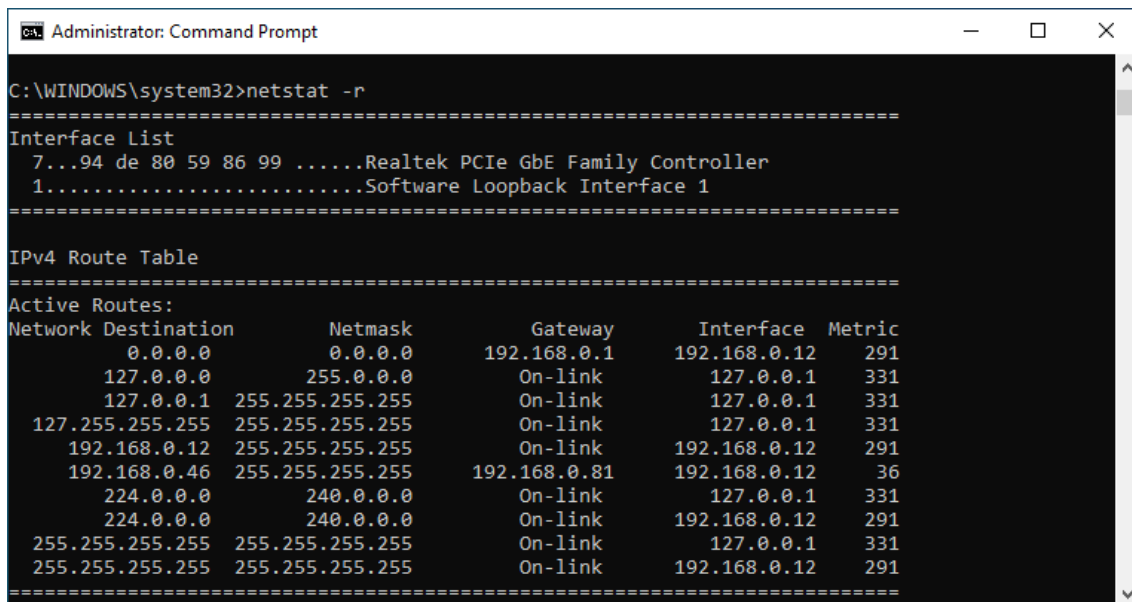


Fig 11: Adding the Static route from source to destination via gateway as NetSim emulation server-192.168.0.81

3. To check whether IP Configuration affected or not type the command as show below
 - a. **netstat -r**



```

Administrator: Command Prompt
C:\WINDOWS\system32>netstat -r

=====
Interface List
  7...94 de 80 59 86 99 .....Realtek PCIe GbE Family Controller
  1.....Software Loopback Interface 1
=====

IPv4 Route Table
=====
Active Routes:
Network Destination        Netmask          Gateway          Interface        Metric
-----
0.0.0.0                    0.0.0.0          192.168.0.1      192.168.0.12     291
127.0.0.0                  255.0.0.0        On-link         127.0.0.1        331
127.0.0.1                  255.255.255.255  On-link         127.0.0.1        331
127.255.255.255            255.255.255.255  On-link         127.0.0.1        331
192.168.0.12               255.255.255.255  On-link         192.168.0.12     291
192.168.0.46               255.255.255.255  192.168.0.81    192.168.0.12     36
224.0.0.0                  240.0.0.0        On-link         127.0.0.1        331
224.0.0.0                  240.0.0.0        On-link         192.168.0.12     291
255.255.255.255            255.255.255.255  On-link         127.0.0.1        331
255.255.255.255            255.255.255.255  On-link         192.168.0.12     291
=====

```

Fig 12 : Display the routing information at source node 192.168.0.12

You can observe that for the node 192.168.0.46, the gateway address assigned is 192.168.0.81 (IP Address of the system where NetSim Emulation server is running)

Results and discussion

After the simulation is completed, you can observe the results using Wireshark captured files. In the Result Dashboard, On the left side, Packet Capture → Emulation and you can see all Emulated Packets captured

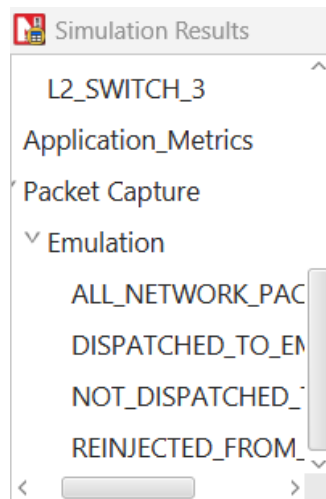


Fig 13: Emulation Packet Capture in Result Dashboard

You can observe original ping traffic generated at the source 192.168.0.12

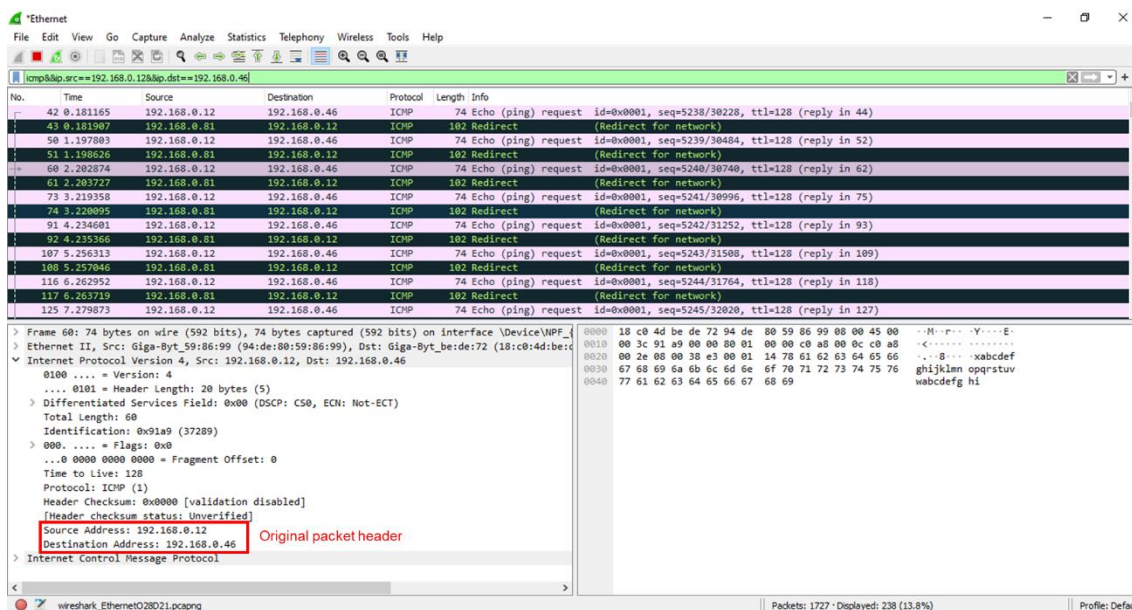


Fig 14: Original ping traffic generated from real source 192.168.0.12, captured using Wireshark.

You can observe false data injected packets in the false destination reply node 192.168.0.68

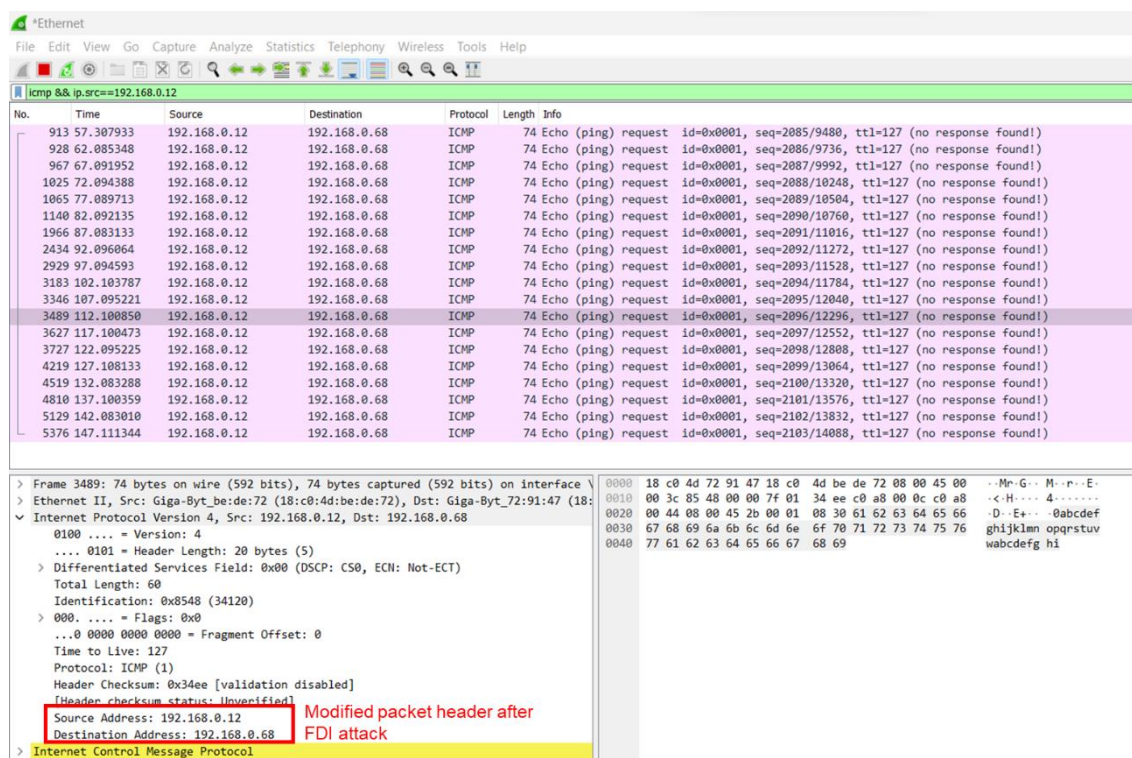


Fig 15: FDI Traffic captured by the destination 192.168.0.68, which is the false data Injected in the ICMP packet header by NetSim

You can observe that the original ping traffic generated by the source 192.168.0.12 to destination 192.168.0.46 has been passed via NetSim Emulation server 192.168.0.81, where we have implemented the FDI attack. After the FDI attack in NetSim will reinject the packet to the actual network with Destination IP modified to 192.168.0.68. You can observe that the real

destination will not receive any ICMP Packets from source 192.168.0.12, Whereas the false destination 192.168.0.68 will receive the icmp traffic from source 192.168.0.12.

Two Types of false data injection attacks: payload modification and header modification

In each of the two cases described earlier, we can model two kinds of attacks:

1. **Packet payload change:** The PING packet by default has its payload as *abcdefghijklmnopqrstuvwabcdefghi*, we modify this to *AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA*
2. **Packet header change.** The destination IP address of the ping is changed from *192.168.0.46* to *192.168.0.68*

Appendix: NetSim source code modifications

MS Visual Studio Development environment is required for editing and building NetSim source codes. Please see this link on setting up Visual Studio

<https://support.tetcos.com/support/solutions/articles/14000138721-what-components-of-visual-studio-community-2022-to-install-and-configure-to-work-with-netsim-source-c>

To open our project source code section, in NetSim home screen to → your work → source code → open code.

NetSim comes with inbuilt low-level functions to capture packets. This code is not open for user modification. The code to access the payload/header and to modify the payload/header is open to users and can be modified. We show below the source code changes we have made in red. Users can alter these functions to implement their own FDI attacks.

Changes to `fn_NetSim_IP_Run()` in `IP.c` file, in `IP` project

Case 1: Payload modification

```
_declspec(dllexport) int fn_NetSim_IP_Run()
{
    //False Data
    char s[BUFSIZ] = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
    switch (pstruEventDetails->nEventType)
    {
        case NETWORK_OUT_EVENT:
        {
            ptrIP_FORWARD_ROUTE route = NULL;
            NetSim_PACKET* packet = pstruEventDetails->pPacket;
            NETWORK_LAYER_PROTOCOL nLocalNetworkProtocol;
            //False Data Injection in Network Layer into packet payload
            if (packet)
            {
                //Device ID of Attacker
                if (pstruEventDetails->nDeviceId == 4){
                    for (int i = 28; i < 60; i++)
                        packet->szPayload->packet[i] = s[i - 28];
                }
            }
            nLocalNetworkProtocol =
            fnGetLocalNetworkProtocol(pstruEventDetails);
            if (nLocalNetworkProtocol)
            {
```

```

        fnCallProtocol(nLocalNetworkProtocol);
        return 0;
    }
    .....
    .....
}
.....
.....
}
}

```

Case 2: Header modification

```

_declspec(dllexport) int fn_NetSim_IP_Run()
{
    //False Data
    char s[BUFSIZ] = "D"; //hexadecimal value for D is 68
    switch (pstruEventDetails->nEventType)
    {
        case NETWORK_OUT_EVENT:
        {
            ptrIP_FORWARD_ROUTE route = NULL;
            NetSim_PACKET* packet = pstruEventDetails->pPacket;
            NETWORK_LAYER_PROTOCOL nLocalNetworkProtocol;
            // False Data Injection in Network Layer into packet header
            if (packet)
            {
                //Device ID of Attacker
                if (pstruEventDetails->nDeviceId == 1){
                    for (int i = 19; i < 20; i++)
                        packet->szPayload->packet[i] = s[i - 19];
                }
            }
            nLocalNetworkProtocol =
fnGetLocalNetworkProtocol(pstruEventDetails);
            if (nLocalNetworkProtocol)
            {
                fnCallProtocol(nLocalNetworkProtocol);
                return 0;
            }
            .....
            .....
        }
        .....
        .....
    }
}

```