False Data Injection Attack

Two cases: Simulation and Emulation
Two Types: Payload modification and Header modification

Software: NetSim Standard v14.1 (64 bit), Visual Studio 2022
Project code download link: https://github.com/NetSim-TETCOS/False-Data-Injection-Attack-in-Internetworks_v14.1/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: FDI implementation within NetSim simulator. Packet payload modification.

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

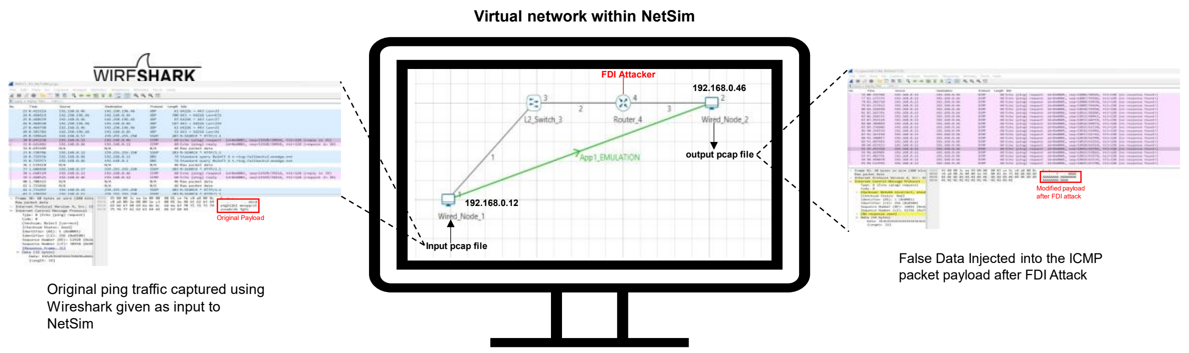
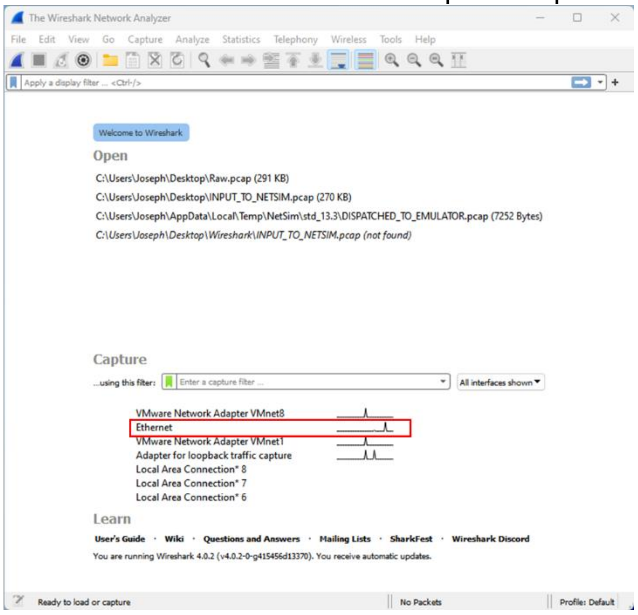

Virtual network within NetSim
WIRESHARK
Original Payload
Original ping traffic captured using Wireshark given as input to NetSim
FDI Attacker
192.168.0.46
L2_Switch_3
Router_4
Wired_Node_2
App1_EMULATION
output pcap file
192.168.0.12
Wired_Node_1
Input pcap file
Modified payload after FDI attack
False Data Injected into the ICMP packet payload after FDI Attack

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.

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.


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Welcome to Wireshark

Open

C:\Users\Joseph\Desktop\Raw.pcap (291 KB)
C:\Users\Joseph\Desktop\INPUT_TO_NETSIM.pcap (270 KB)
C:\Users\Joseph\AppData\Local\Temp\NetSim\std_13.3\DISPATCHED_TO_EMULATOR.pcap (7252 Bytes)
C:\Users\Joseph\Desktop\Wireshark\INPUT_TO_NETSIM.pcap (not found)

Capture

...using this filter: Enter a capture filter ... All interfaces shown

VMware Network Adapter VMnet8
Ethernet
VMware Network Adapter VMnet1
Adapter for loopback traffic capture
Local Area Connection* 8
Local Area Connection* 7
Local Area Connection* 6

Learn

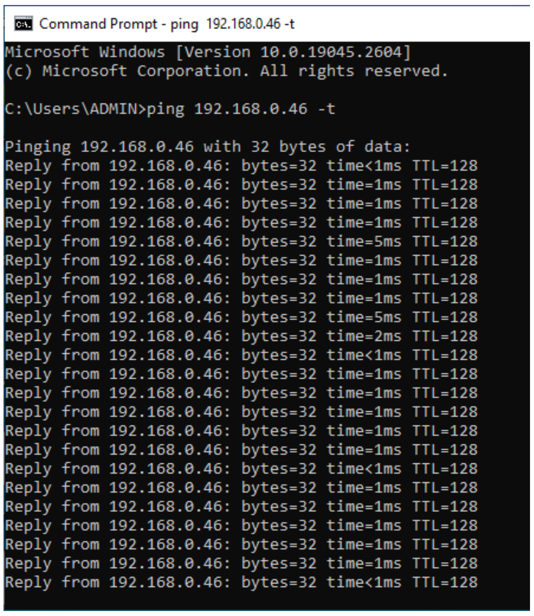
User's Guide · Wiki · Questions and Answers · Mailing Lists · SharkFest · Wireshark Discord

You are running Wireshark 4.0.2 (v4.0.2-0-g415456d13370). You receive automatic updates.

Ready to load or capture | No Packets | Profile: Default

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

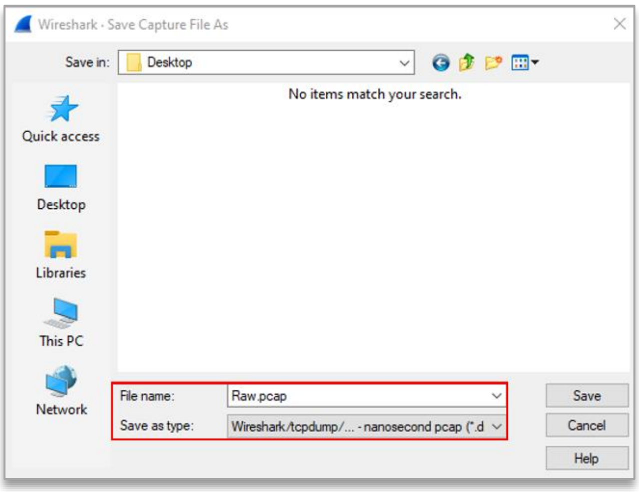
3. In this Example we have considered areal source with 192.168.0.12 and a real destination with IP 192.168.0.46. Open command line at source device and enter the command
   - ping 192.168.0.46 -t



```
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 you have captured the desired ping traffic stop the Wireshark packet capture using stop option 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**.

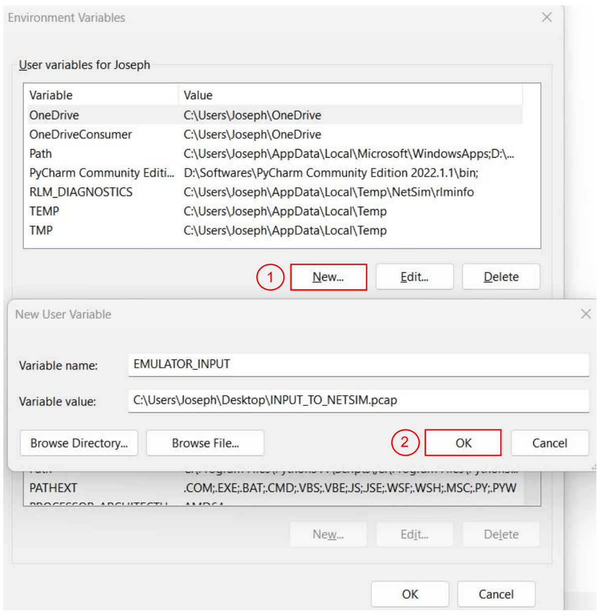


File name: Raw.pcap
Save as type: Wireshark/tcpdump/... - nanosecond pcap (*.d

5. This PCAP file 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 by providing pcap packet capture file as input to NetSim

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



Environment Variables

User variables for Joseph

Variable	Value
OneDrive	C:\Users\Joseph\OneDrive
OneDriveConsumer	C:\Users\Joseph\OneDrive
Path	C:\Users\Joseph\AppData\Local\Microsoft\WindowsApps;D:\...
PyCharm Community Editi...	D:\Softwares\PyCharm Community Edition 2022.1.1\bin;
RLM_DIAGNOSTICS	C:\Users\Joseph\AppData\Local\Temp\NetSim\rlminfo
TEMP	C:\Users\Joseph\AppData\Local\Temp
TMP	C:\Users\Joseph\AppData\Local\Temp

New User Variable

Variable name: EMULATOR_INPUT

Variable value: C:\Users\Joseph\Desktop\INPUT_TO_NETSIM.pcap

PATHEXT .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.PYW

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-

Implementing the FDI attack

1. Run the NetSim in Administrator Mode (Right Click on NetSim Icon → Run as Administrator)
2. The **FDI_Attack_in_Internetworks_v14.1** 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.

3. The saved network scenario consists of
   - 2 Wired Node
   - 1 L2 Switch
   - 1 Router

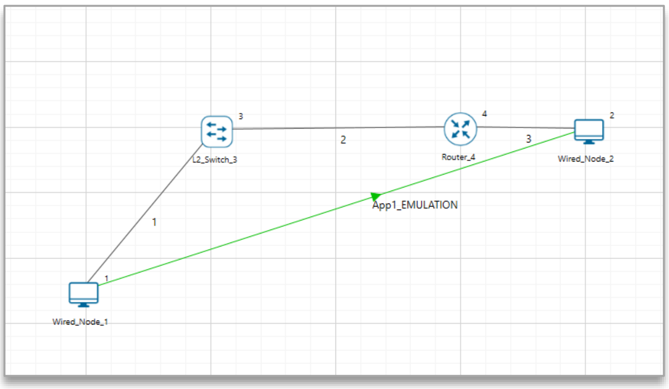


Fig 5: NetSim Emulation Scenario, Wired_Node_1 device mapped for Source IP 192.168.0.12 and Wired_Node_2 device mapped for Destination IP 192.168.0.46

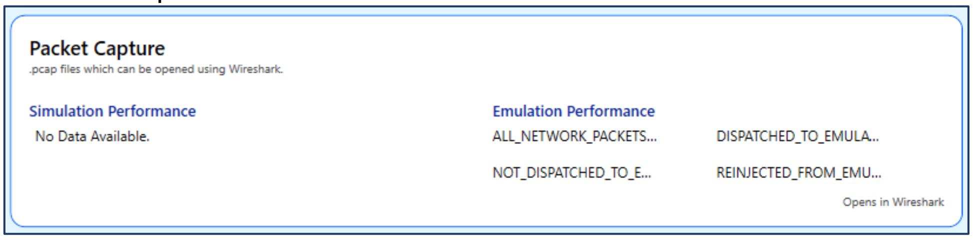
4. Application Properties
   - Application Type - EMULATION
   - Source IP - 192.168.0.12

   - Destination IP – 192.168.0.46
5. Run the Simulation for 100 sec.

Note: The source IP address refers to the IP address of the system from which you are initiating the ping command.
The destination IP address to the IP address of the device or system that you are pinging.

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.



Packet Capture
.pcap files which can be opened using Wireshark.

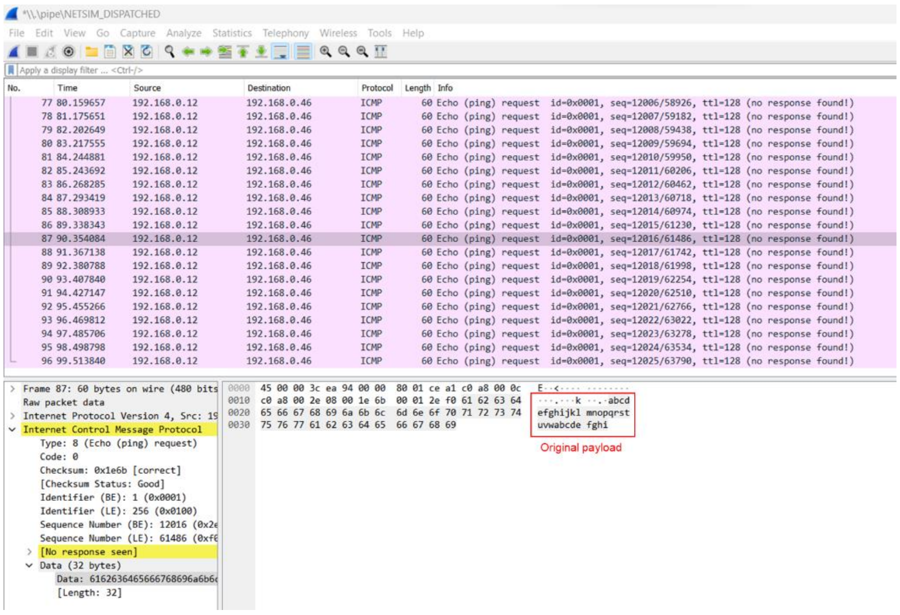
Simulation Performance
No Data Available.

Emulation Performance
ALL_NETWORK_PACKETS...
DISPATCHED_TO_EMULA...
NOT_DISPATCHED_TO_E...
REINJECTED_FROM_EMU...

Opens in Wireshark

Fig 6: Emulation Packet Capture in Result Dashboard

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


*\\.\pipe\NETSIM_DISPATCHED
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.    Time       Source         Destination    Protocol Length Info
    77 80.159657  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12006/58926, ttl=128 (no response found!)
    78 81.175651  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12007/59182, ttl=128 (no response found!)
    79 82.202649  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12008/59438, ttl=128 (no response found!)
    80 83.217555  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12009/59694, ttl=128 (no response found!)
    81 84.244881  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12010/59950, ttl=128 (no response found!)
    82 85.243692  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12011/60206, ttl=128 (no response found!)
    83 86.268285  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12012/60462, ttl=128 (no response found!)
    84 87.293419  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12013/60718, ttl=128 (no response found!)
    85 88.308933  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12014/60974, ttl=128 (no response found!)
    86 89.338343  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12015/61230, ttl=128 (no response found!)
    87 90.354084  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12016/61486, ttl=128 (no response found!)
    88 91.367138  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12017/61742, ttl=128 (no response found!)
    89 92.380788  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12018/61998, ttl=128 (no response found!)
    90 93.407840  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12019/62254, ttl=128 (no response found!)
    91 94.427147  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12020/62510, ttl=128 (no response found!)
    92 95.455266  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12021/62766, ttl=128 (no response found!)
    93 96.469812  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12022/63022, ttl=128 (no response found!)
    94 97.485706  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12023/63278, ttl=128 (no response found!)
    95 98.498798  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12024/63534, ttl=128 (no response found!)
    96 99.513840  192.168.0.12   192.168.0.46   ICMP       60 Echo (ping) request  id=0x0001, seq=12025/63790, ttl=128 (no response found!)
```

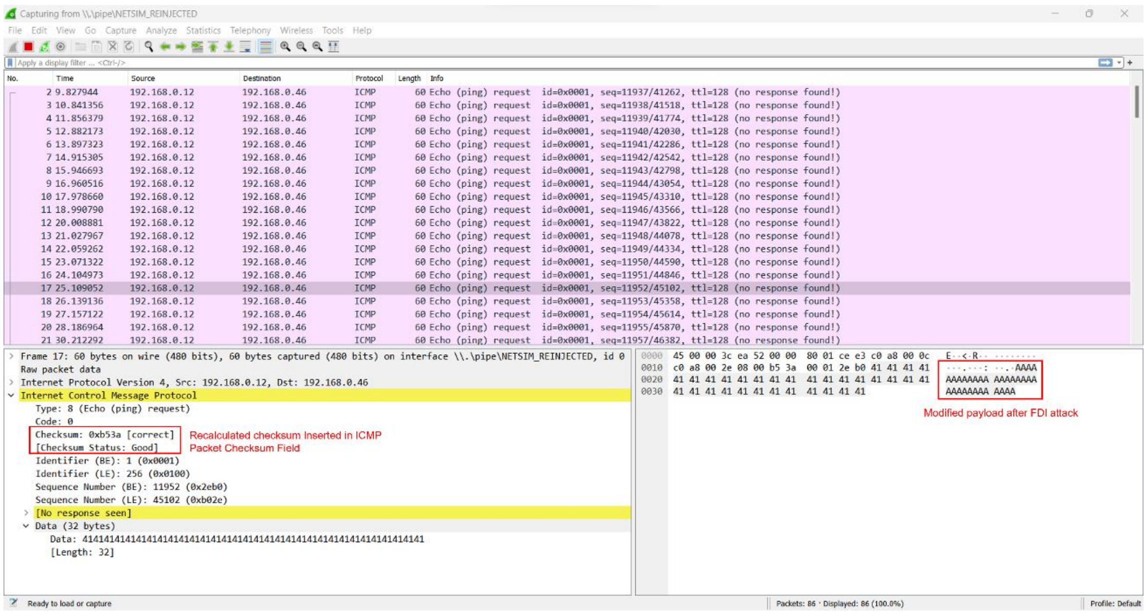
```
> Frame 87: 60 bytes on wire (480 bits
  Raw packet data
> Internet Protocol Version 4, Src: 19
v Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x1e6b [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence Number (BE): 12016 (0x2e
    Sequence Number (LE): 61486 (0xf0
  > [No response seen]
  v Data (32 bytes)
      Data: 6162636465666768696a6b6c
      [Length: 32]
```

```
0000  45 00 00 3c ea 94 00 00  80 01 ce a1 c0 a8 00 0c   E··<···· ········
0010  c0 a8 00 2e 08 00 1e 6b  00 01 2e f0 61 62 63 64   ···.···k ···.abcd
0020  65 66 67 68 69 6a 6b 6c  6d 6e 6f 70 71 72 73 74   efghijkl mnopqrst
0030  75 76 77 61 62 63 64 65  66 67 68 69               uvwabcde fghi
```

Fig 7: Original payload captured by NetSim emulator

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

Note: *You should select the any ICMP Packet to observe the changes.*



Capturing from \\.\pipe\NETSIM_REINJECTED
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.   Time        Source         Destination    Protocol  Length Info
    2 9.827944    192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11937/41262, ttl=128 (no response found!)
    3 10.841356   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11938/41518, ttl=128 (no response found!)
    4 11.856379   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11939/41774, ttl=128 (no response found!)
    5 12.882173   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11940/42030, ttl=128 (no response found!)
    6 13.897323   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11941/42286, ttl=128 (no response found!)
    7 14.915305   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11942/42542, ttl=128 (no response found!)
    8 15.946693   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11943/42798, ttl=128 (no response found!)
    9 16.960516   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11944/43054, ttl=128 (no response found!)
   10 17.978660   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11945/43310, ttl=128 (no response found!)
   11 18.990790   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11946/43566, ttl=128 (no response found!)
   12 20.008881   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11947/43822, ttl=128 (no response found!)
   13 21.027967   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11948/44078, ttl=128 (no response found!)
   14 22.059262   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11949/44334, ttl=128 (no response found!)
   15 23.071322   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11950/44590, ttl=128 (no response found!)
   16 24.104973   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11951/44846, ttl=128 (no response found!)
   17 25.109052   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11952/45102, ttl=128 (no response found!)
   18 26.139136   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11953/45358, ttl=128 (no response found!)
   19 27.157122   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11954/45614, ttl=128 (no response found!)
   20 28.186964   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11955/45870, ttl=128 (no response found!)
   21 30.212292   192.168.0.12   192.168.0.46   ICMP      60 Echo (ping) request  id=0x0001, seq=11957/46382, ttl=128 (no response found!)
```

```
> Frame 17: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface \\.\pipe\NETSIM_REINJECTED, id 0
  Raw packet data
> Internet Protocol Version 4, Src: 192.168.0.12, Dst: 192.168.0.46
v Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xb53a [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence Number (BE): 11952 (0x2eb0)
    Sequence Number (LE): 45102 (0xb02e)
  > [No response seen]
  v Data (32 bytes)
      Data: 4141414141414141414141414141414141414141414141414141414141414141
      [Length: 32]
```

Recalculated checksum Inserted in ICMP Packet Checksum Field

```
0000  45 00 00 3c ea 52 00 00  80 01 ce e3 c0 a8 00 0c   E··<·R·· ········
0010  c0 a8 00 2e 08 00 b5 3a  00 01 2e b0 41 41 41 41   ···.···: ··.·AAAA
0020  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41   AAAAAAAA AAAAAAAA
0030  41 41 41 41 41 41 41 41  41 41 41 41               AAAAAAAA AAAA
```

Modified payload after FDI attack

Fig 8: Traffic with false data injected. Observe the difference in payload and checksum is recalculated and inserted in ICMP packet checksum field.

Case 2: FDI implementation in NetSim emulator. Packet header modification.

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

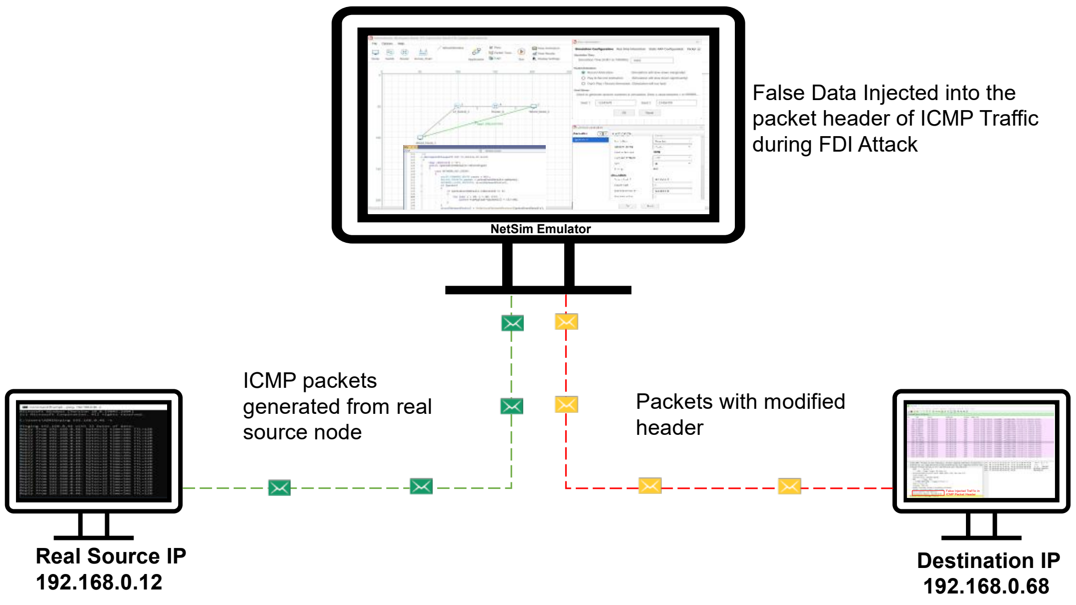

False Data Injected into the packet header of ICMP Traffic during FDI Attack
NetSim Emulator
ICMP packets generated from real source node
Packets with modified header
Real Source IP 192.168.0.12
Destination IP 192.168.0.68

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

Note: Ensure that the environmental variables do not contain an entry for **EMULATOR_INPUT** in either the user or system variables.

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
    - 2 Wired Node
    - 1 L2 Switch
    - 1 Router

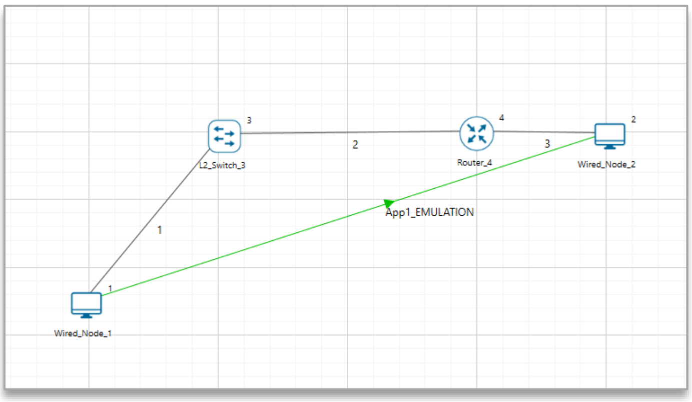


Fig 10: NetSim Emulation Scenario, Wired_Node_1 device mapped for Source IP 192.168.0.12 and Wired_Node_2 device mapped for Destination IP 192.168.0.46

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

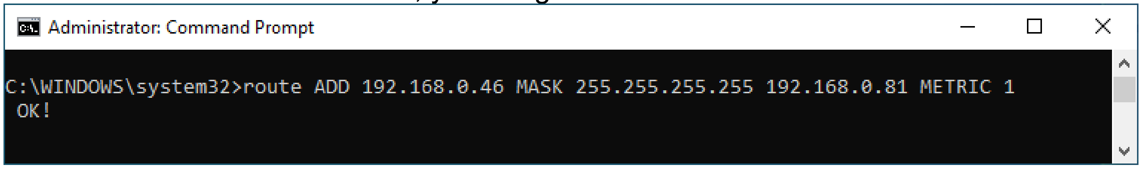
Note: The source IP address refers to the IP address of the system from which you are initiating the ping command.
The destination IP address to the IP address of the device or system that you are pinging.

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

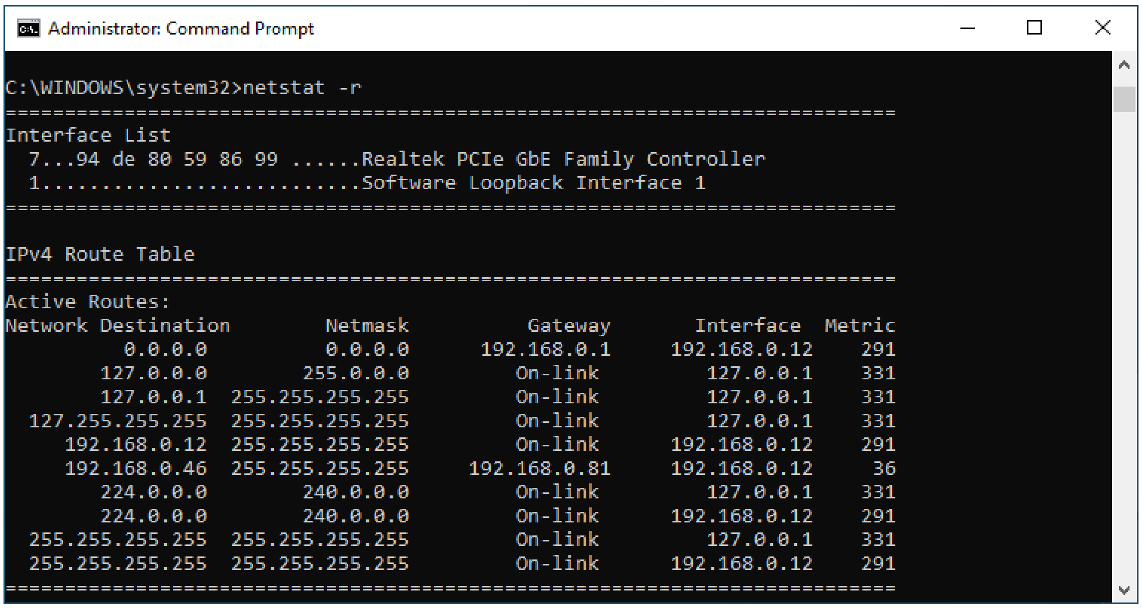
1. Open command prompt in Administrator Mode
2. Type the command.
    - route add <Network Destination> mask <Subnet Mask> <Gateway IP> metric 1
    - **route add 192.168.0.46 mask 255.255.255.0 192.168.0.81 metric 1**
    - After the Execution , you will get "OK".



```
Administrator: Command Prompt

C:\WINDOWS\system32>route ADD 192.168.0.46 MASK 255.255.255.255 192.168.0.81 METRIC 1
 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
    - **netstat -r**

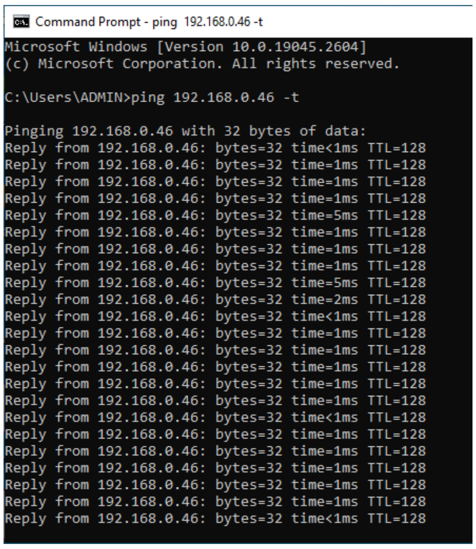


```
Administrator: Command Prompt

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 of routing information at source node 192.168.0.12

You can observe that for the Destination 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)

8. Open command line at Source node 192.168.0.12 and enter the command.
    - ping 192.168.0.46 -t



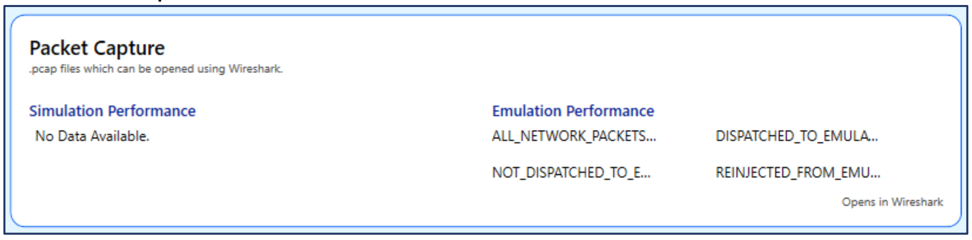
Command Prompt - ping 192.168.0.46 -t

```
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 13 : Pinging to destination IP 192.168.0.46

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.



Packet Capture
.pcap files which can be opened using Wireshark.

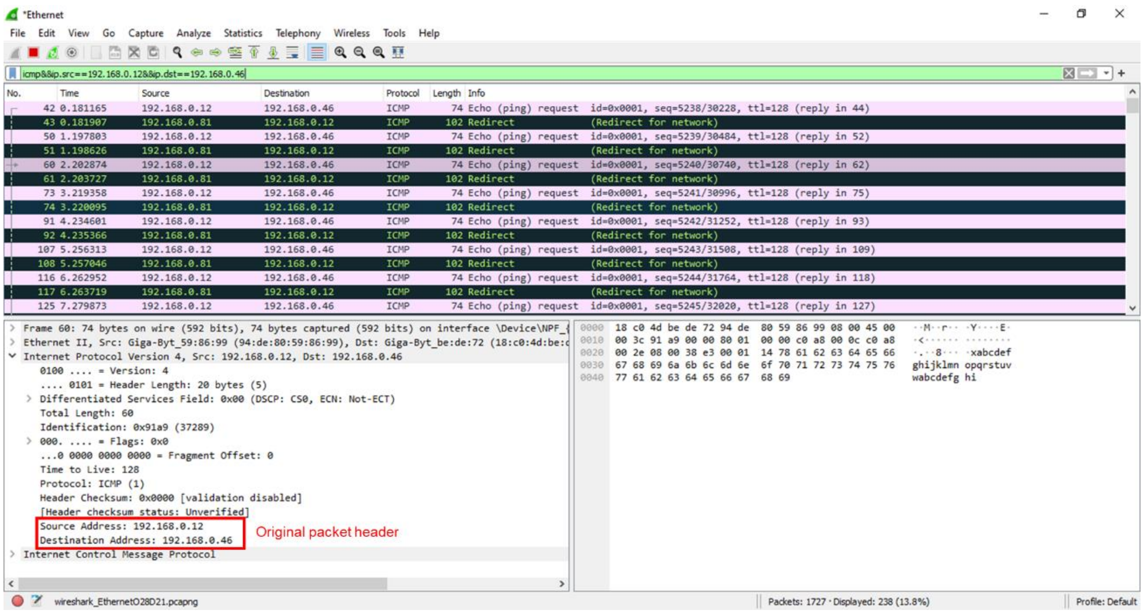
Simulation Performance
No Data Available.

Emulation Performance
ALL_NETWORK_PACKETS...   DISPATCHED_TO_EMULA...
NOT_DISPATCHED_TO_E...   REINJECTED_FROM_EMU...

Opens in Wireshark

Fig 14: Emulation Packet Capture in Result Dashboard

We can observe original ping traffic generated at the source 192.168.0.12



icmp&&ip.src==192.168.0.12&&ip.dst==192.168.0.46

```
No.   Time       Source         Destination    Protocol Length Info
   42 0.181165   192.168.0.12   192.168.0.46   ICMP      74 Echo (ping) request  id=0x0001, seq=5238/30228, ttl=128 (reply in 44)
   43 0.181907   192.168.0.81   192.168.0.12   ICMP     102 Redirect             (Redirect for network)
   50 1.197803   192.168.0.12   192.168.0.46   ICMP      74 Echo (ping) request  id=0x0001, seq=5239/30484, ttl=128 (reply in 52)
   51 1.198626   192.168.0.81   192.168.0.12   ICMP     102 Redirect             (Redirect for network)
   60 2.202874   192.168.0.12   192.168.0.46   ICMP      74 Echo (ping) request  id=0x0001, seq=5240/30740, ttl=128 (reply in 62)
   61 2.203727   192.168.0.81   192.168.0.12   ICMP     102 Redirect             (Redirect for network)
   73 3.219358   192.168.0.12   192.168.0.46   ICMP      74 Echo (ping) request  id=0x0001, seq=5241/30996, ttl=128 (reply in 75)
   74 3.220095   192.168.0.81   192.168.0.12   ICMP     102 Redirect             (Redirect for network)
   91 4.234601   192.168.0.12   192.168.0.46   ICMP      74 Echo (ping) request  id=0x0001, seq=5242/31252, ttl=128 (reply in 93)
   92 4.235366   192.168.0.81   192.168.0.12   ICMP     102 Redirect             (Redirect for network)
  107 5.256313   192.168.0.12   192.168.0.46   ICMP      74 Echo (ping) request  id=0x0001, seq=5243/31508, ttl=128 (reply in 109)
  108 5.257046   192.168.0.81   192.168.0.12   ICMP     102 Redirect             (Redirect for network)
  116 6.262952   192.168.0.12   192.168.0.46   ICMP      74 Echo (ping) request  id=0x0001, seq=5244/31764, ttl=128 (reply in 118)
  117 6.263719   192.168.0.81   192.168.0.12   ICMP     102 Redirect             (Redirect for network)
  125 7.279873   192.168.0.12   192.168.0.46   ICMP      74 Echo (ping) request  id=0x0001, seq=5245/32020, ttl=128 (reply in 127)
```

```
> Frame 60: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{
> Ethernet II, Src: Giga-Byt_59:86:99 (94:de:80:59:86:99), Dst: Giga-Byt_be:de:72 (18:c0:4d:be:d
v Internet Protocol Version 4, Src: 192.168.0.12, Dst: 192.168.0.46
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0x91a9 (37289)
  > 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: ICMP (1)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.0.12
    Destination Address: 192.168.0.46
> Internet Control Message Protocol
```

Original packet header

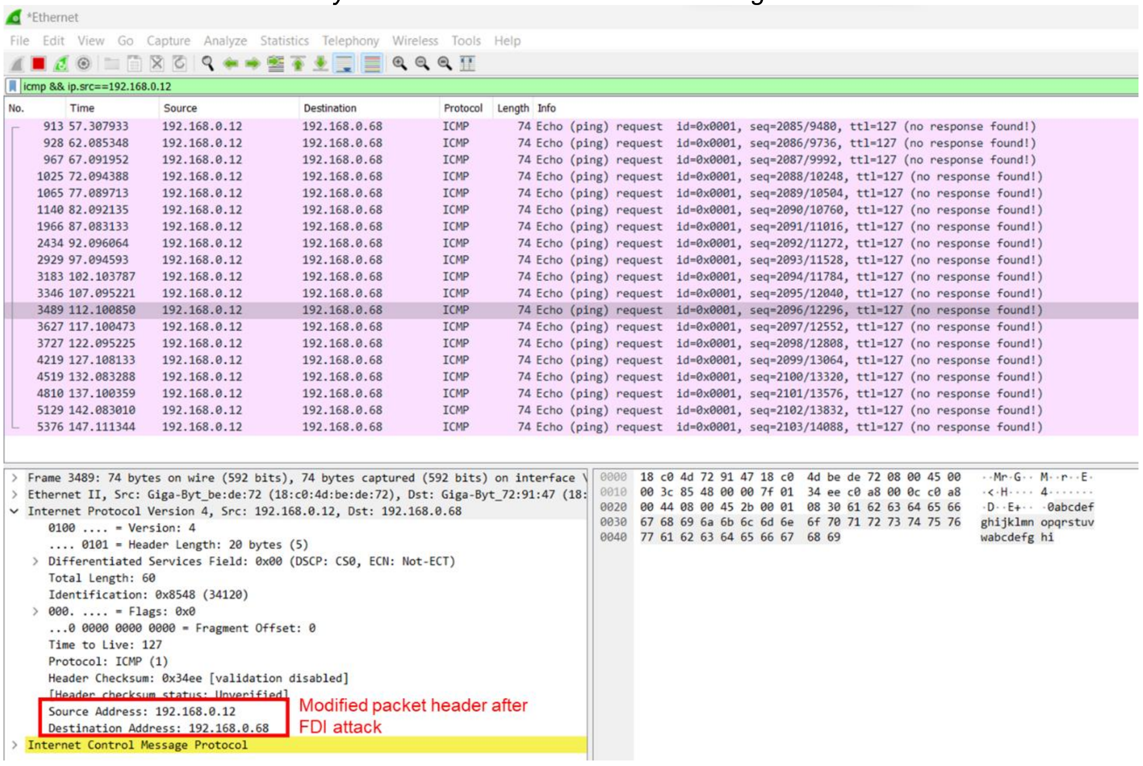
```
0000  18 c0 4d be de 72 94 de  80 59 86 99 08 00 45 00   ··M··r·· ·Y····E·
0010  00 3c 91 a9 00 00 80 01  00 00 c0 a8 00 0c c0 a8   ·<······ ········
0020  00 2e 08 00 38 e3 00 01  14 78 61 62 63 64 65 66   ·.··8··· ·xabcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi
```

wireshark_EthernetO28D21.pcapng | Packets: 1727 · Displayed: 238 (13.8%) | Profile: Default

Fig 15: Original ICMP traffic generated from real source 192.168.0.12, captured using Wireshark.

We can observe false data injected packets in the false destination node 192.168.0.68

Note: *You should select the any ICMP Packet to observe the changes.*



```
icmp && ip.src==192.168.0.12

No.   Time          Source         Destination    Protocol  Length Info
  913 57.307933     192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2085/9480, ttl=127 (no response found!)
  928 62.085348     192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2086/9736, ttl=127 (no response found!)
  967 67.091952     192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2087/9992, ttl=127 (no response found!)
 1025 72.094388     192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2088/10248, ttl=127 (no response found!)
 1065 77.089713     192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2089/10504, ttl=127 (no response found!)
 1140 82.092135     192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2090/10760, ttl=127 (no response found!)
 1966 87.083133     192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2091/11016, ttl=127 (no response found!)
 2434 92.096064     192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2092/11272, ttl=127 (no response found!)
 2929 97.094593     192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2093/11528, ttl=127 (no response found!)
 3183 102.103787    192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2094/11784, ttl=127 (no response found!)
 3346 107.095221    192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2095/12040, ttl=127 (no response found!)
 3489 112.100850    192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2096/12296, ttl=127 (no response found!)
 3627 117.100473    192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2097/12552, ttl=127 (no response found!)
 3727 122.095225    192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2098/12808, ttl=127 (no response found!)
 4219 127.108133    192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2099/13064, ttl=127 (no response found!)
 4519 132.083288    192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2100/13320, ttl=127 (no response found!)
 4810 137.100359    192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2101/13576, ttl=127 (no response found!)
 5129 142.083010    192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2102/13832, ttl=127 (no response found!)
 5376 147.111344    192.168.0.12   192.168.0.68   ICMP      74 Echo (ping) request  id=0x0001, seq=2103/14088, ttl=127 (no response found!)

> Frame 3489: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \
> Ethernet II, Src: Giga-Byt_be:de:72 (18:c0:4d:be:de:72), Dst: Giga-Byt_72:91:47 (18:
v Internet Protocol Version 4, Src: 192.168.0.12, Dst: 192.168.0.68
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0x8548 (34120)
  > 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 127
    Protocol: ICMP (1)
    Header Checksum: 0x34ee [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.0.12
    Destination Address: 192.168.0.68
> Internet Control Message Protocol

0000  18 c0 4d 72 91 47 18 c0  4d be de 72 08 00 45 00   ··Mr·G·· M··r··E·
0010  00 3c 85 48 00 00 7f 01  34 ee c0 a8 00 0c c0 a8   ·<·H···· 4·······
0020  00 44 08 00 45 2b 00 01  08 30 61 62 63 64 65 66   ·D··E+·· ·0abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi
```

*Ethernet
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Modified packet header after FDI attack

Fig 16: FDI Traffic captured by the destination 192.168.0.68, which is the false data Injected in the ICMP packet header by NetSim.

We can observe that the original ping traffic generated by the source 192.168.0.12 destined to 192.168.0.46 was passed via NetSim Emulation server 192.168.0.81. At the NetSim Emulation server we implemented the FDI attack. After the FDI attack in NetSim will reinject the modified 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, since the destination address in different. If there is a machine with IP 192.168.0.68 in the network, then that machine will now 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. Once the code changes done rebuild the project by right click on IP project→ Rebuild, Once you rebuild is successful the project code modification will be affected in NetSim.

Case 1: Payload modification

Add a new function before **fn_NetSim_IP_Run()** and after **ip_handle_processing_delay() in IP.c file, in IP project.**

```
static void ip_handle_processing_delay()
{
        .......
}

// Function to calculate the Internet Checksum
uint16_t calculateChecksum(const uint8_t* data, size_t length) {
        uint32_t sum = 0;

        // Process each 16-bit chunk of data
        while (length > 1) {
                sum += ((uint16_t)data[0] << 8) + data[1];
                data += 2;
                length -= 2;
        }

        // If there's a remaining odd byte, add it to the sum
        if (length == 1) {
                sum += ((uint16_t)data[0] << 8);
        }

        // Fold the 32-bit sum to a 16-bit checksum
        while (sum >> 16) {
                sum = (sum & 0xFFFF) + (sum >> 16);
        }

        // Return the one's complement of the final sum
        return (uint16_t)(~sum);
}

//Seperate into 2 Bytes
static void separateBytes(uint16_t value, uint8_t* highByte, uint8_t* lowByte) {
        *highByte = (uint8_t)(value >> 8);  // Get the high byte
        *lowByte = (uint8_t)(value & 0xFF); // Get the low byte
}

/**
This function is called by NetworkStack.dll, whenever the event gets triggered
```

```
inside the NetworkStack.dll for IP.It includes NETWORK_OUT,NETWORK_IN and
TIMER_EVENT.
*/
_declspec(dllexport) int fn_NetSim_IP_Run()
{
        .......
}
```

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

```
_declspec(dllexport) int fn_NetSim_IP_Run()
{
        //False Data
        char s[BUFSIZ] = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
        uint8_t checkszero = 0x00;
        uint8_t packet_icmp[40];

        switch (pstruEventDetails->nEventType)
        {
                case NETWORK_OUT_EVENT:
                {
                        ptrIP_FORWARD_ROUTE route = NULL;
                        NetSim_PACKET* packet = pstruEventDetails->pPacket;
                        NETWORK_LAYER_PROTOCOL nLocalNetworkProtcol;
                        //False Data Injection in Network Layer into packet payload and
                        regenerate the checksum field.
                        if (packet)
                        {
                                //Device ID of Attacker
                                if (pstruEventDetails->nDeviceId == 4)
                                {
                                        for (int i = 28; i < 60; i++)
                                                packet->szPayload->packet[i] = s[i - 28];
                                                //Modifying the payload by inserting False Data
                                }
                                //Checksum Recalculation
                                if (pstruEventDetails->nDeviceId == 4)
                                {
                                        // Read the packet data into a C array.
                                        unsigned char* packet_data = (unsigned char*)packet-
>szPayload->packet;

                                        //Extract the ICMP Packet Payload
                                        for (int k = 20; k < 60; k++) {
                                                packet_icmp[k - 20] = (uint8_t)packet_data[k];
                                        }

                                        //Set the Checksum Variable to 0 while calculating the
                                        checksum
                                        packet_icmp[2] = checkszero;
                                        packet_icmp[3] = checkszero;

                                        //Calculate the new checksum value for ICMP Packet
                                        Payload
                                        size_t length = sizeof(packet_icmp);
                                        uint16_t checksum = calculateChecksum(packet_icmp,
length);

                                        //Separate the 16-bit value to two 8-bit values
                                        uint8_t highByte, lowByte;
                                        separateBytes(checksum, &highByte, &lowByte);

                                        //Update the checksum value in checksum field
                                        packet->szPayload->packet[22] = highByte;
                                        packet->szPayload->packet[23] = lowByte;
                                }
                        }
                        nLocalNetworkProtcol                                              =
fnGetLocalNetworkProtocol(pstruEventDetails);
                        if (nLocalNetworkProtcol)
                        {
                                fnCallProtocol(nLocalNetworkProtcol);
                                return 0;
                        }

                ………
                ………
                }
        ……….
        ……….
        }
}
```

Case 2: Header modification

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

```
_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 nLocalNetworkProtcol;
                        // 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];
                                }
                        }
```

```
                              nLocalNetworkProtcol =
fnGetLocalNetworkProtocol(pstruEventDetails);
                              if (nLocalNetworkProtcol)
                              {
                                        fnCallProtocol(nLocalNetworkProtcol);
                                        return 0;
                              }
                    ………
                    ………
                    }
          ……….
          ……….
          }
}
```