Dos Attack in Internet of Things

Software Used: NetSim Standard v11.1 (32/64 bit), Visual Studio 2019

A Denial of Service (DoS) attack is an attempt to make a system unavailable to the intended user(s), such as preventing access to a website. A successful DoS attack consumes all available network or system resources, usually resulting in a slowdown or server crash. Whenever multiple sources are coordinating in the DoS attack, it becomes known as a DDoS (Distributed Denial of Service) attack.

Standard DDoS Attack types:

1. SYN Flood
2. UDP Flood
3. SMBLoris
4. ICMP Flood
5. HTTP GET Flood

SYN Flood:

TCP SYN floods are DoS attacks that attempt to flood the DNS server with new TCP connection requests. Normally, a client initiates a TCP connection through a three way handshake of messages:

- The client requests a connection by sending a SYN (synchronize) message to the server.
- The server acknowledges the request by sending SYN-ACK back to the client.
- The client answers with a responding ACK, establishing the connection.
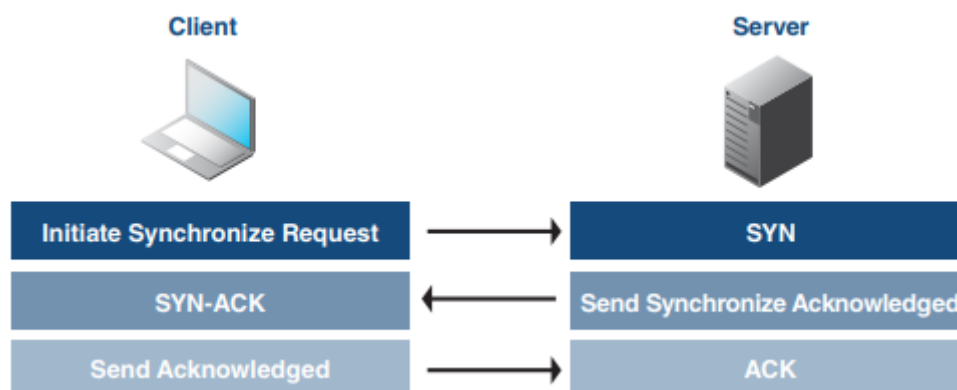
This triple exchange is the foundation for every connection established using the Transmission Control Protocol (TCP). A SYN Flood is one of the most common forms of DDoS attacks. It occurs when an attacker sends a succession of TCP Synchronize (SYN) requests to the target in an attempt to consume enough resources to make the server unavailable for legitimate users. This works because a SYN request opens network communication between a prospective client and the target server. When the server receives a SYN request, it responds acknowledging the request and holds the communication open while it waits for the client to acknowledge the open connection. However, in

a successful SYN Flood, the client acknowledgment never arrives, thus consuming the server's resources until the connection times out. A large number of incoming SYN requests to the target server exhausts all available server resources and results in a successful DoS attack.

Before implementing this project in NetSim, users have to understand the steps given below:

1. **TCP Log file**
   - User need to understand the TCP log file which will get created in the temp path of NetSim <Windows Temp Folder>/NetSim>

   - The TCP Log file is usually a very large file and hence is disabled by default in NetSim.

   - To enable logging, go to TCP.c inside the TCP project and change the function bool isTCPlog() to return true instead of false.

2. **At malicious node:**

   Create a new timer event called SYN_FLOOD in TCP for sending TCP_SYN packets that should be triggered for every 1000 micro seconds. This will create and send the TCP_SYN packet for every 1000 micro seconds. SYN request opens network communication between a client and the target

3. **At Target node:**

   When the target receives a SYN request, it responds acknowledging the request and holds the communication open while it waits for the client to acknowledge the open connection. If a SYN packet arrives at Receiver, it should reply with a SYN_ACK packet. For this SYN_ACK packet, add a processing time of 2000 micro seconds in Ethernet Physical Out. This delays the arrival of SYN_ACK at source node. During this delay, another SYN packet will get created at the malicious node. A large number of incoming SYN requests to the target exhausts all available server resources and results in a successful DoS attack

SYN_FLOOD in NetSim:

To implement this project in NetSim, we have created SYN_FLOOD.c file inside TCP project. The file contains the following functions:

- int socket_creation();

   This function is used to create a new socket and update the socket parameters
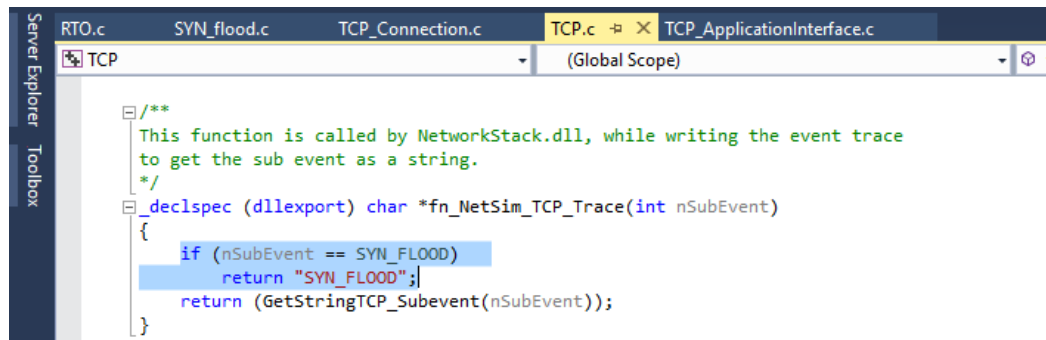
- static void send_syn_packet(PNETSIM_SOCKET s);

   This function is used to create and send SYN packet to the network layer

- void syn_flood();

This function is used to check whether the socket is present or not and also adds a timer event called SYN_FLOOD (triggers for every 1000µs)
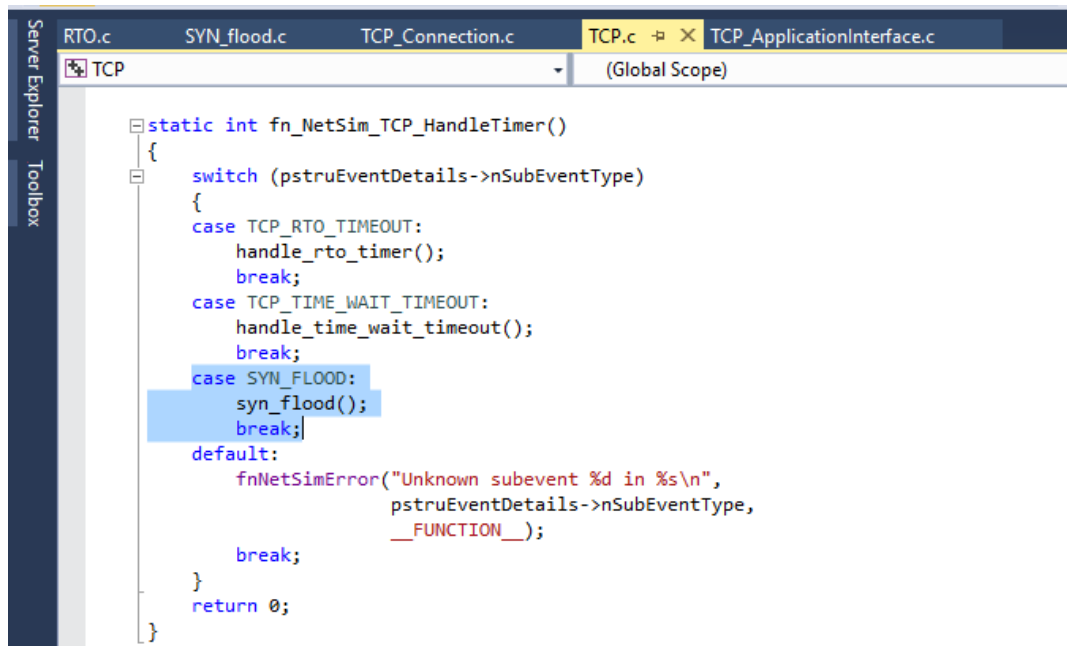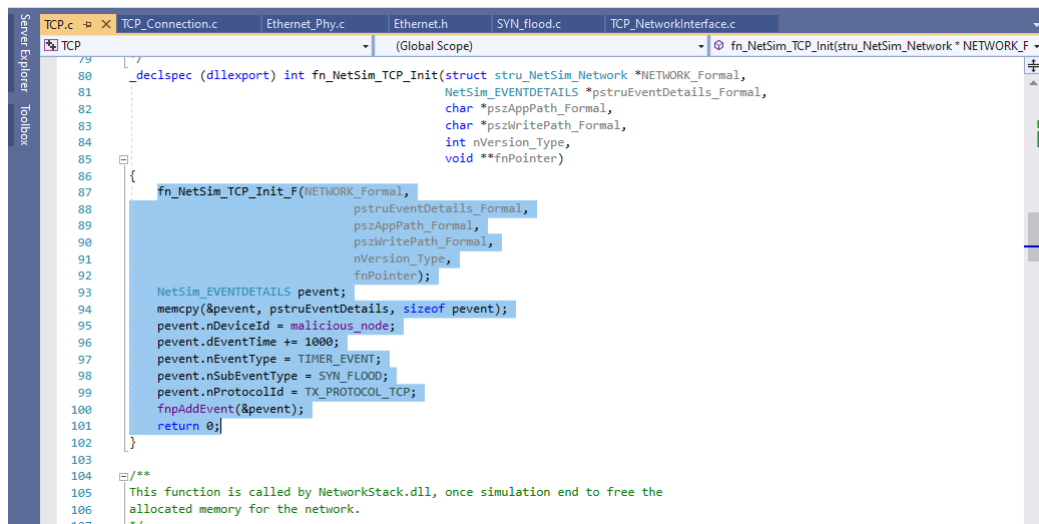
Code modifications done in NetSim:

1. We have added the following lines of code in fn_NetSim_TCP_Trace() function present in TCP.c file inside TCP project. This is used to add the SYN_FLOOD sub-events in Event Trace file

```
RTO.c        SYN_flood.c        TCP_Connection.c        TCP.c  ⊕ ✕  TCP_ApplicationInterface.c
⊕ TCP                                              ▾        (Global Scope)                              ▾ ⊕ f

    /**
    This function is called by NetworkStack.dll, while writing the event trace
    to get the sub event as a string.
    */
    _declspec (dllexport) char *fn_NetSim_TCP_Trace(int nSubEvent)
    {
        if (nSubEvent == SYN_FLOOD)
            return "SYN_FLOOD";
        return (GetStringTCP_Subevent(nSubEvent));
    }
```

2. We have added the following lines of code in fn_NetSim_TCP_HandleTimer() function present in TCP.c file inside TCP project. Used to add a TCP sub_event called SYN_FLOOD

```
RTO.c        SYN_flood.c        TCP_Connection.c        TCP.c  ⊕ ✕  TCP_ApplicationInterface.c
⊕ TCP                                              ▾        (Global Scope)                              ▾

    static int fn_NetSim_TCP_HandleTimer()
    {
        switch (pstruEventDetails->nSubEventType)
        {
        case TCP_RTO_TIMEOUT:
            handle_rto_timer();
            break;
        case TCP_TIME_WAIT_TIMEOUT:
            handle_time_wait_timeout();
            break;
        case SYN_FLOOD:
            syn_flood();
            break;
        default:
            fnNetSimError("Unknown subevent %d in %s\n",
                        pstruEventDetails->nSubEventType,
                        __FUNCTION__);
            break;
        }
        return 0;
    }
```

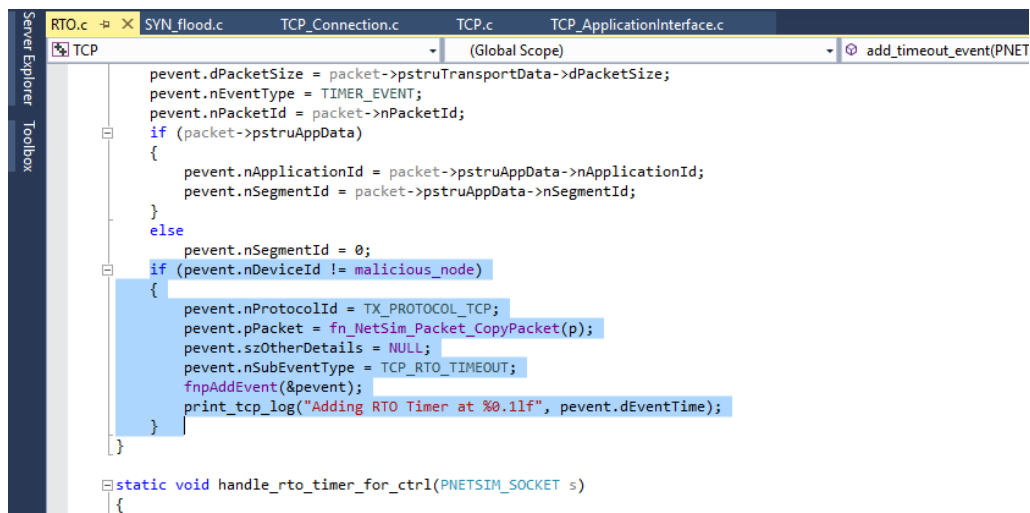3. And modified the following lines of code in fn_NetSim_TCP_Init() function resent in TCP.c inside TCP project

4. And modified the following lines of code in add_timeout_event() present in RTO.c file inside TCP project which avoids RTO timer for malicious nodes
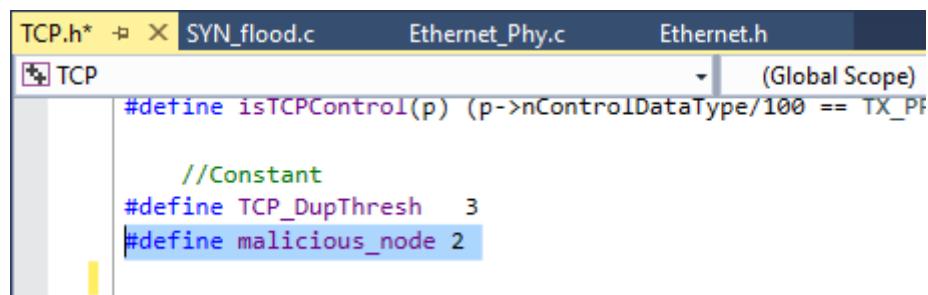

5. Users can give their own malicious ID in **TCP.h** file inside TCP project

6. Users can give their own target ID in **SYN_FLOOD.c** file inside TCP project

7. Added the following line in TCP_Enum.h file inside TCP project to add a new TCP_subevent called SYN_FLOOD

8. SYN_FLOOD.c file contains the following functions

```
SYN_flood.c ⊕ ×
TCP                                    ▾   (Global Scope)

      ⊟static void send_syn_packet(PNETSIM_SOCKET s)
       {
            NetSim_PACKET* syn = create_syn(s, pstruEventDetails->dEventTime);

            s->tcb->SND.UNA = s->tcb->ISS;
            s->tcb->SND.NXT = s->tcb->ISS + 1;
            tcp_change_state(s, TCPCONNECTION_SYN_SENT);

            s->tcb->synRetries++;

            s->tcpMetrics->synSent++;

            send_to_network(syn, s);
            add_timeout_event(s, syn);
       }
```

```
Ethernet_Phy.c      SYN_flood.c ⊕ ×  TCP_NetworkInterface.c
TCP                              ▾    (Global Scope)              ▾
  76    }
  77
  78    ⊟static PNETSIM_SOCKET socket_creation()
  79     {
  80          static int s_id = 100;
  81          ptrSOCKETINTERFACE sId = (ptrSOCKETINTERFACE)pstruEventDetails->szOtherDetails;
  82          PNETSIM_SOCKET newSocket = tcp_create_socket();
  83
  84          add_to_socket_list(malicious_node, newSocket);
  85
  86          PSOCKETADDRESS localsocketAddr = (PSOCKETADDRESS)calloc(1, sizeof * localsocketAddr);
  87          localsocketAddr->ip = DEVICE_NWADDRESS(malicious_node, 1);
  88          localsocketAddr->port = 0;
  89
  90          PSOCKETADDRESS remotesocketAddr = (PSOCKETADDRESS)calloc(1, sizeof * remotesocketAddr);
  91          remotesocketAddr->ip = DEVICE_NWADDRESS(target_node, 1);
  92          remotesocketAddr->port = 0;
  93
  94          newSocket->SocketId = s_id;
  95          s_id++;
  96
  97          newSocket->localAddr = localsocketAddr;
  98          newSocket->remoteAddr = remotesocketAddr;
  99
 100          newSocket->localDeviceId = malicious_node;
 101          newSocket->remoteDeviceId = target_node;
 102
 103          newSocket->sId = sId;
 104
 105          return newSocket;
 106    }
 107
98 %   ▾    ⊘ No issues found
```

9. Added PROCESSING_TIME macro in Ethernet.h file inside ETHERNET project

```
Ethernet.h ⊕ ×  Ethernet_Phy.c      SYN_flood.c      TCP_NetworkInterface.c
Ethernet                                     ▾        (Global Scope)              ▾
  28
  29      #define ETH_IFG 0.960 //Micro sec
  30
  31      #define Processing_TIME 3000
  32
```

10. Modified the following lines of code in fn_NetSim_Ethernet_HandlePhyOut() function present in Ethernet_Phy.c file inside Ethernet project.

```
        if (!packet)
            return 2; // No packet is there for transmission

        double start;

        if (pstruEventDetails->nDeviceId == target_node && (packet->nControlDataType == 40102 || packet->nControlDataType == 40105))
        {
            if (phy->lastPacketEndTime + phy->IFG <= pstruEventDetails->dEventTime)
                start = pstruEventDetails->dEventTime + Processing_TIME;
            else
                start = phy->lastPacketEndTime + phy->IFG + Processing_TIME;
        }
        else
        {
            if (phy->lastPacketEndTime + phy->IFG <= pstruEventDetails->dEventTime)
                start = pstruEventDetails->dEventTime;
            else
                start = phy->lastPacketEndTime + phy->IFG;
        }
```
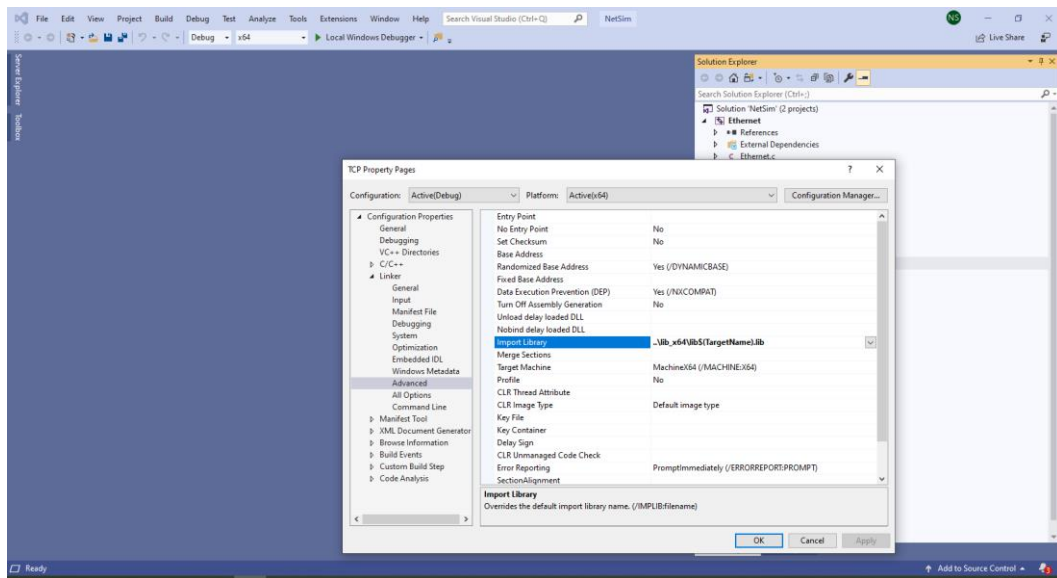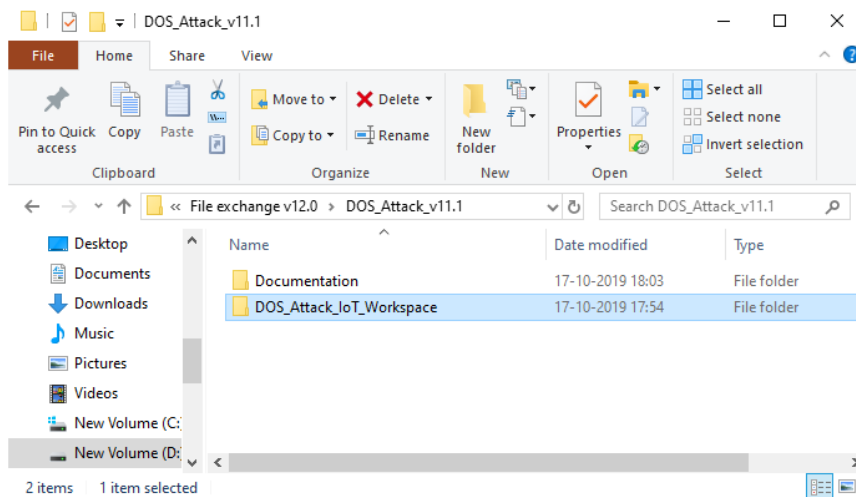
11. Right click on TCP project → Properties→Linker → Advanced →import library 32-bit and 64-bit **..\lib\lib$(TargetName).lib** or **..\lib_x64\lib$(TargetName).lib**
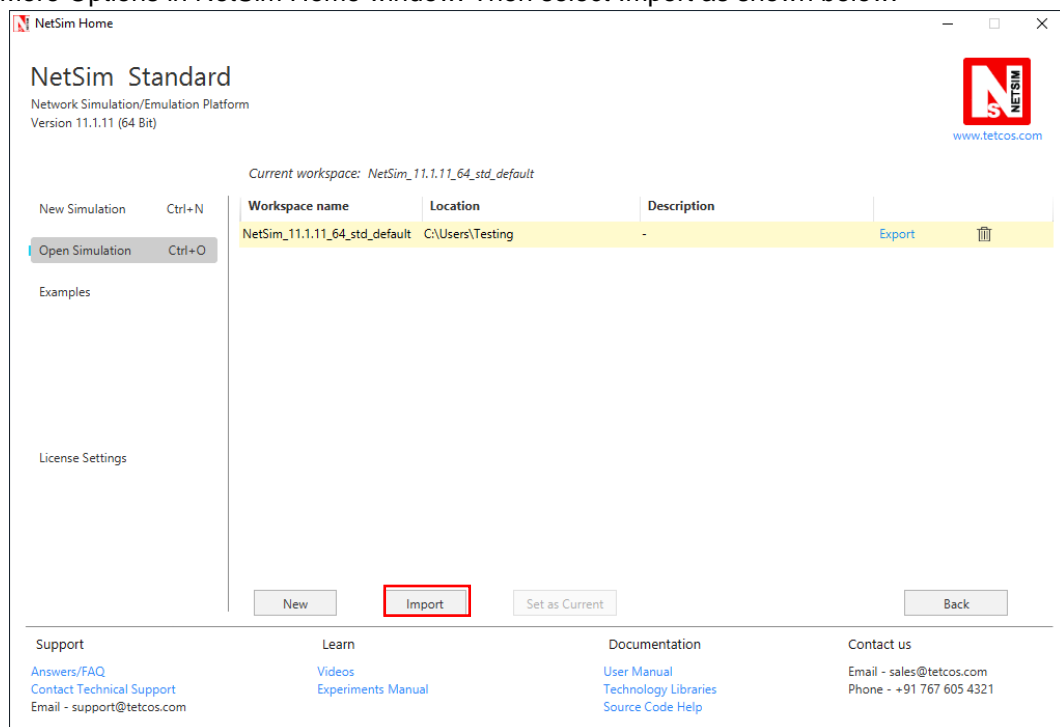

Steps:

1. The downloaded project folder contains the folders Documentation, and DOS_Attack_IoT_Workspace directory as shown below:

2. Import DOS_Attack_IoT_Workspace by going to Open Simulation->Workspace Options->More Options in NetSim Home window. Then select Import as shown below:
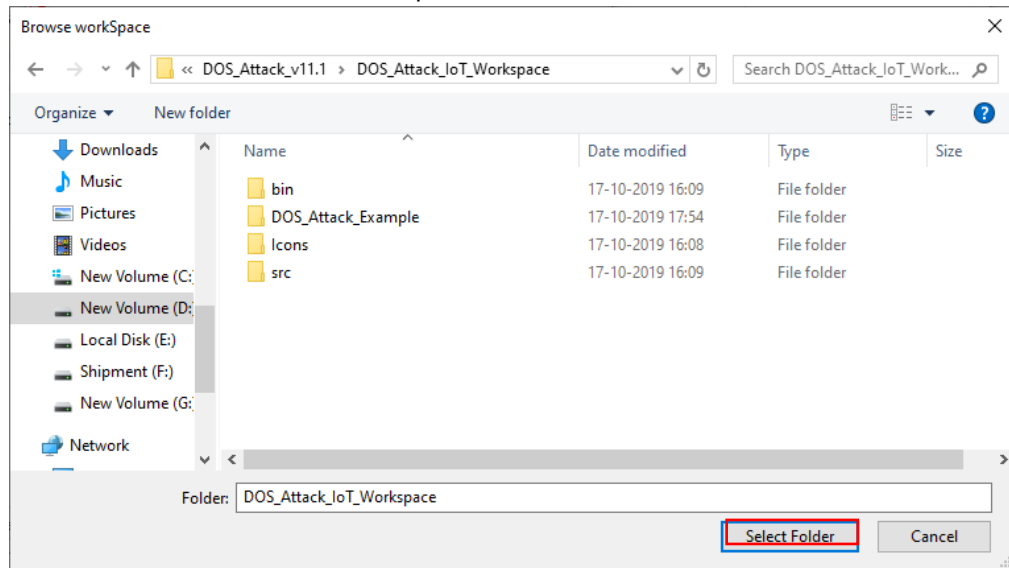

3. It displays a window where users need to give the path of the workspace folder and click on OK as shown below:
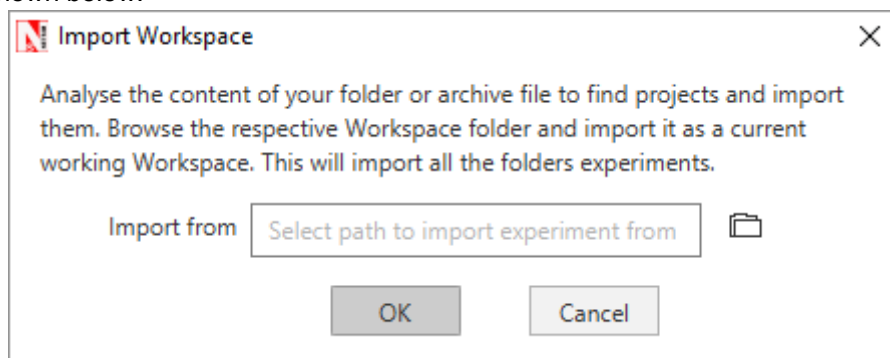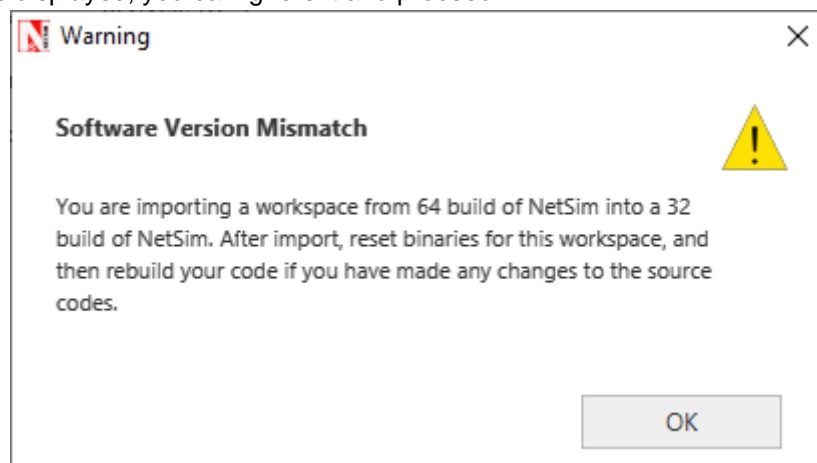
4. Browse to the DOS_Attack_IoT_Workspace folder and click on select folder as shown below:

5. After this click on OK button in the Import Workspace window.

6. While importing the workspace, if the following warning message indicating Software Version Mismatch is displayed, you can ignore it and proceed.


7. The Imported workspace will be set as the current workspace automatically. To see the imported workspace, click on Open Simulation->Workspace Options->More Options as shown below:
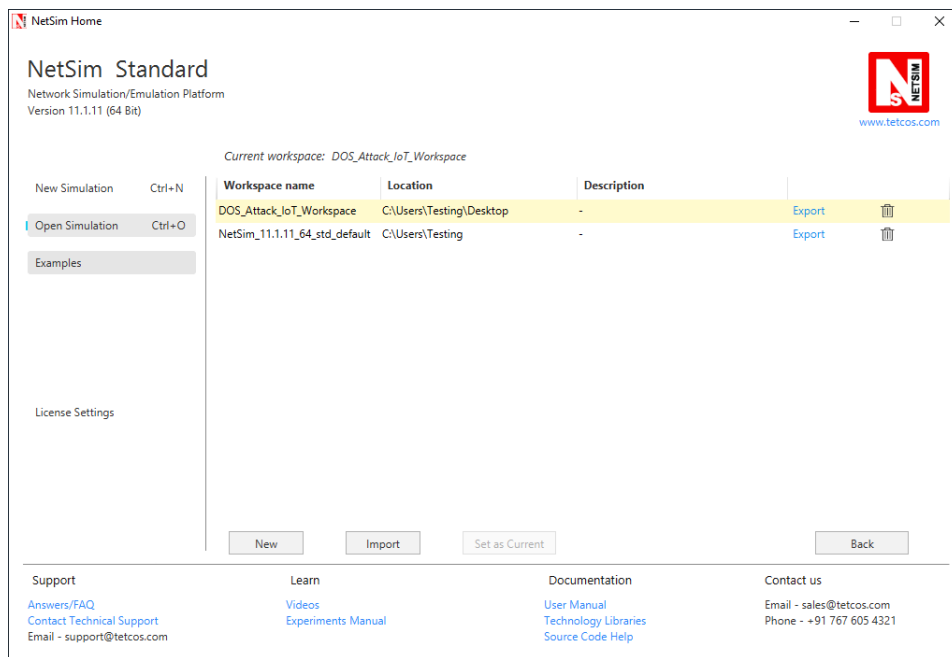
8. Open the Source codes in Visual Studio by going to Open Simulation-> Workspace Options and Clicking on Open code button as shown below:
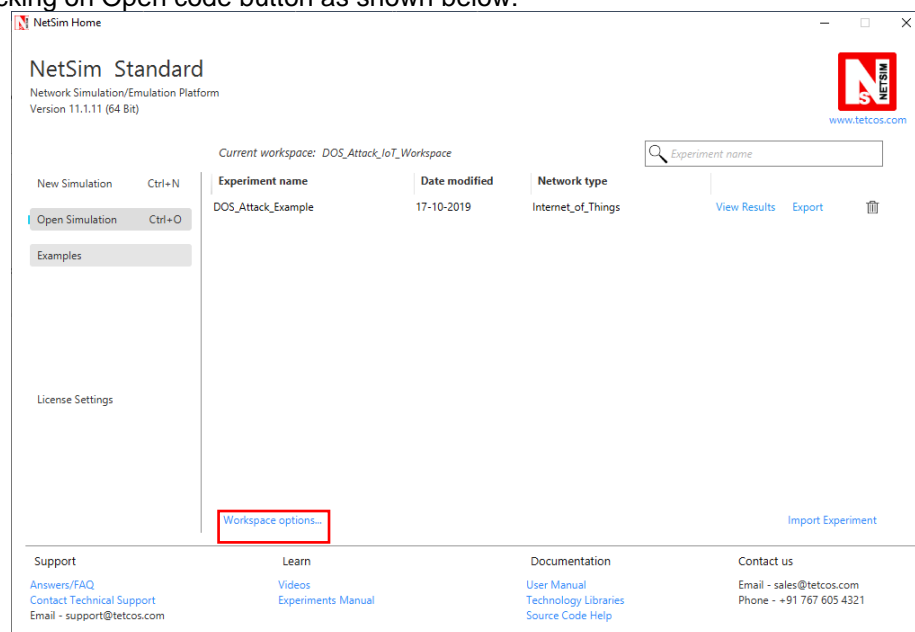

9. Under the **TCP** project in the solution explorer you will be able to see that **SYN_FLOOD.c** file.
10. Based on whether you are using NetSim 32 bit or 64 bit setup you can configure Visual studio to build 32 bit or 64 bit Dll files respectively as shown below:
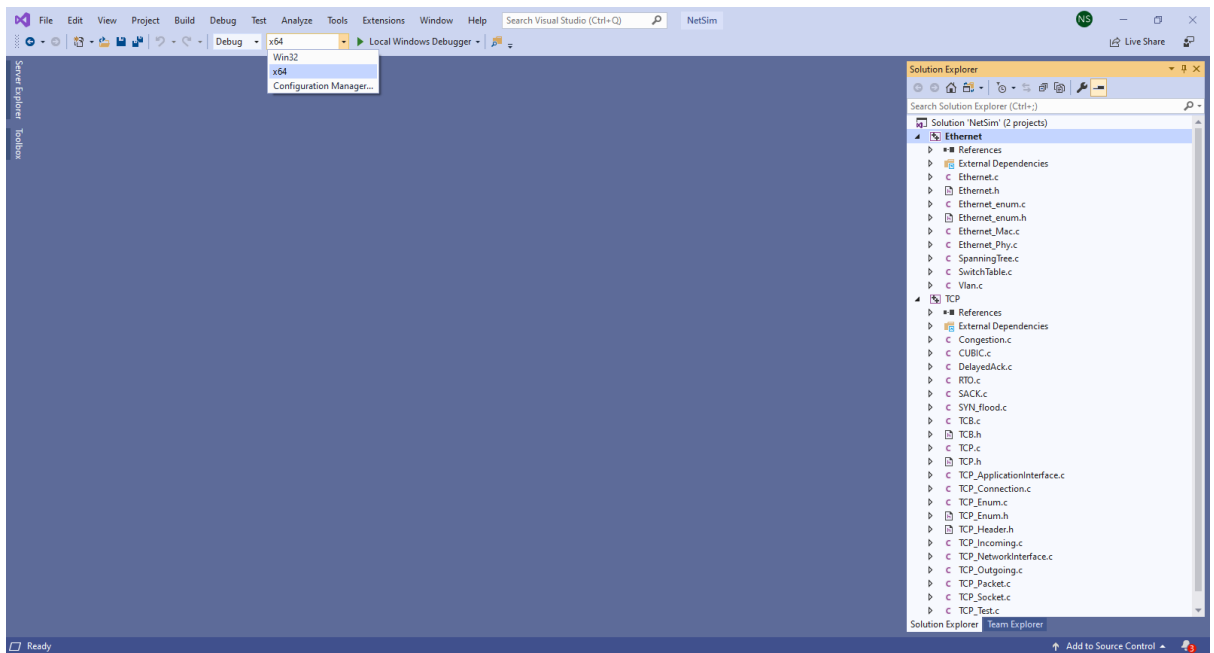
11. Right click on the solution in the solution explorer and select Rebuild.
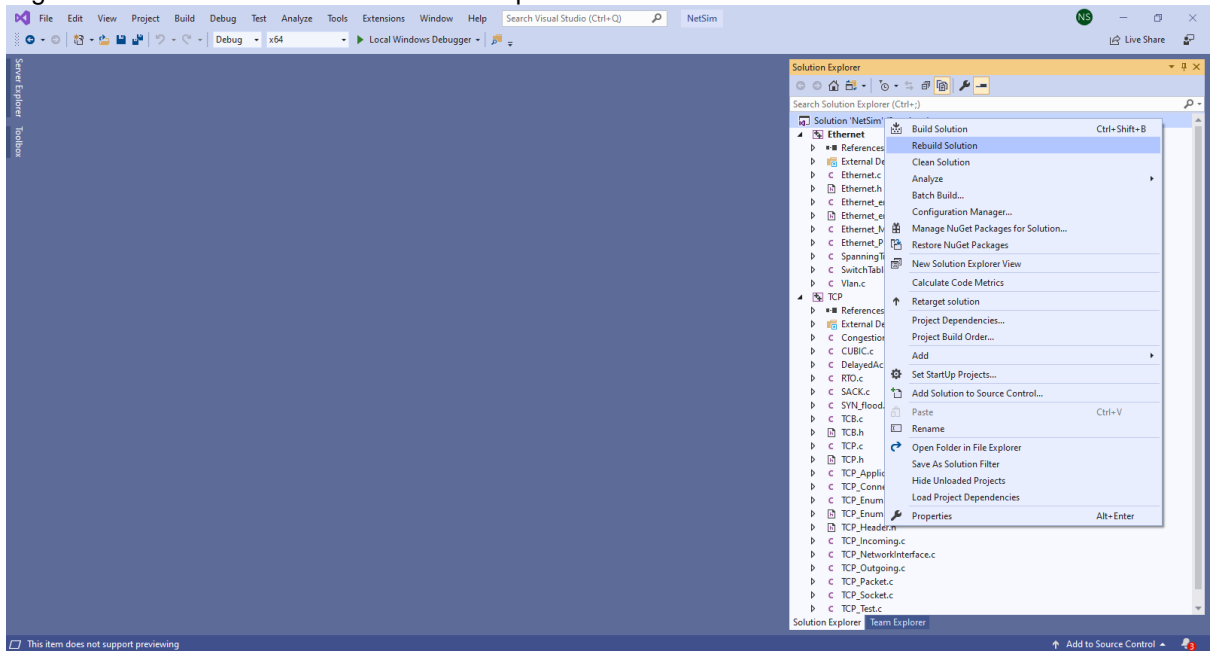
12. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries.

13. Run NetSim as Administrative mode.

14. Then DOS_Attack_IoT_Workspace comes with a sample configuration that is already saved. To open this example, go to Open Simulation and click on the DOS_Attack_Example that is present under the list of experiments as shown below:
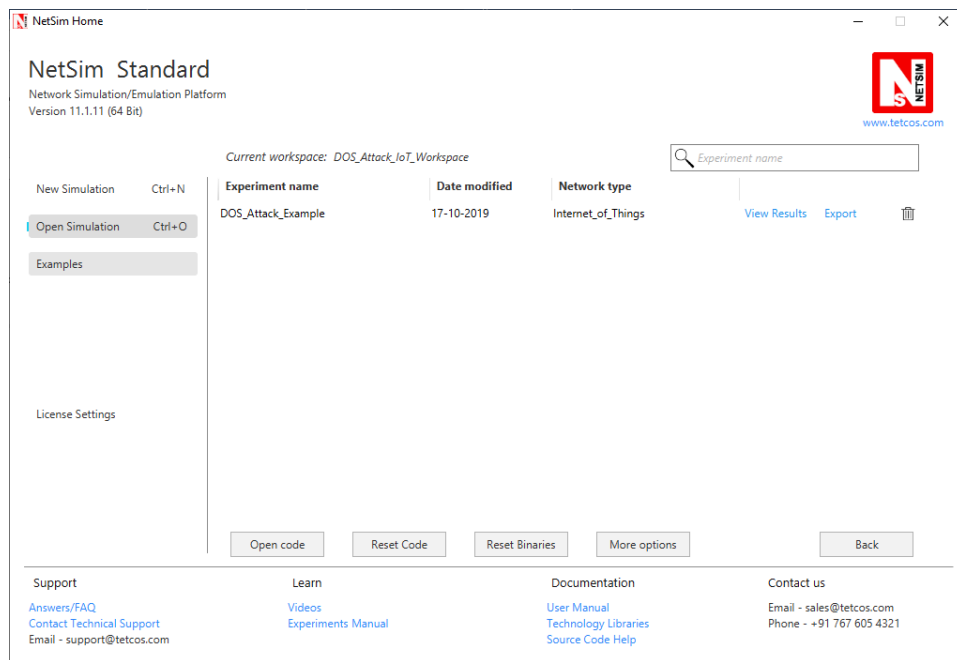
15. The saved network scenario consisting of 2 sensors, 1 6LOWPAN Gateway, 1 router, and 1 wired node in the grid environment forming a IoT Network. Traffic is configured from sensor node to the Wired Node.
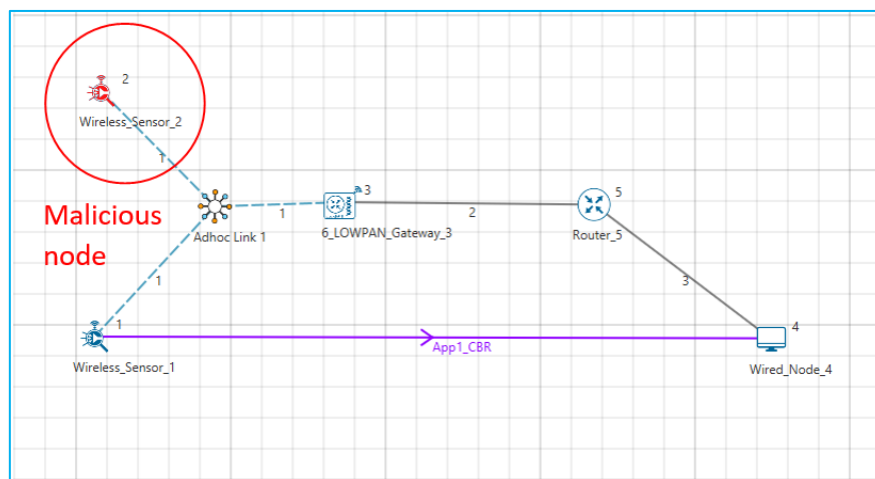
16. Run the simulation for 100 seconds.

Result:

After simulation, open metrics window and observe the Application_Throughput is 0. Go to animation window and observe, the application packets are reaching to the destination but from the destination source is not getting the acknowledgement back, so the source is sending the same packet multiple times.

Note: Users can also create their own network scenarios in Internet of Things and run simulation.