Dos Attack in Internetworks

Software Used: NetSim Standard v13.1 64 bit, Visual Studio 2019

Project Download Link:

https://github.com/NetSim- https://github.com/NetSim-TETCOS/DOS_Attack_Internetworks_v13.1/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:

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:**

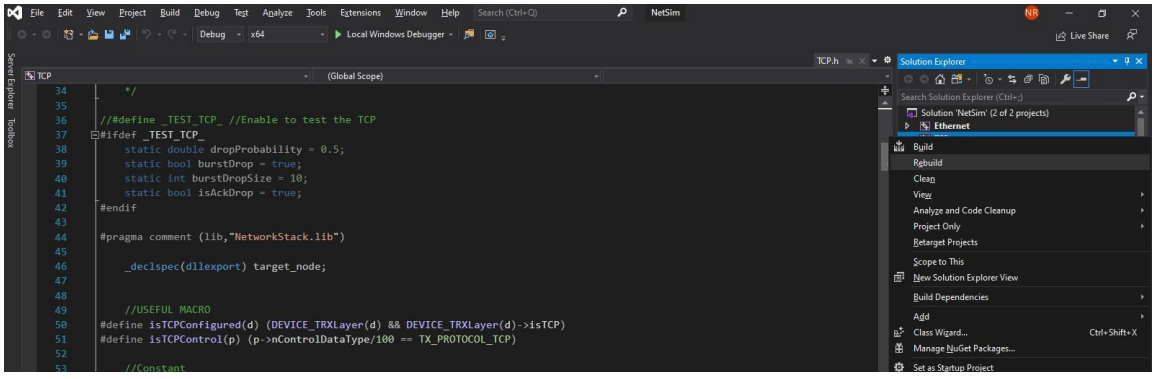
C functions for the SYN_FLOOD attack

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

- int is_malicious_node();//This function is used to check the node is malicious node or not
- 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)

Steps to simulate the attack

1. Open the Source codes in Visual Studio by going to Your work-> Source code option and Clicking on Open code in NetSim Home Screen window.
2. In Visual Studio under the **TCP** project in the solution explorer you will be able to see that **SYN_FLOOD.c** file.
3. Right click on the solution in the solution explorer and select Rebuild (Note: first rebuild the TCP project and then rebuild the Ethernet project

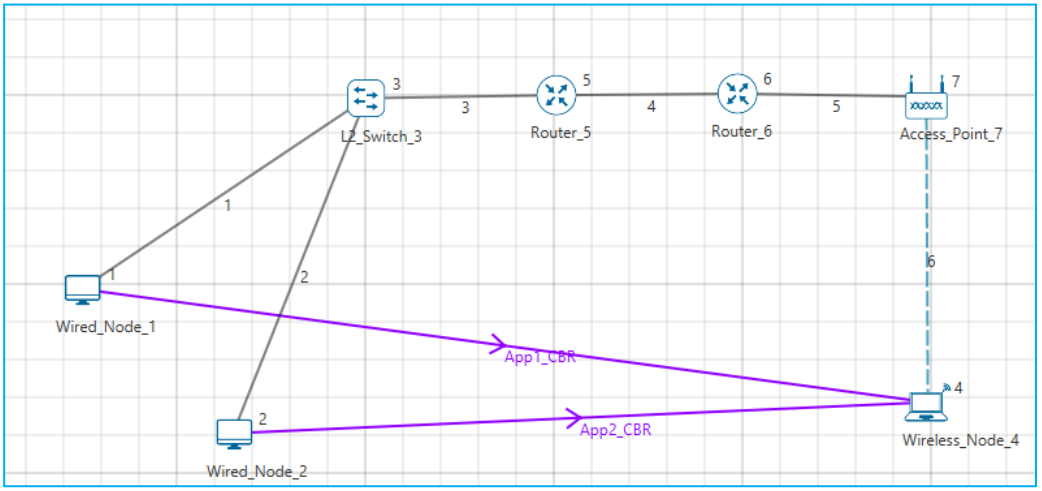


4. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries.

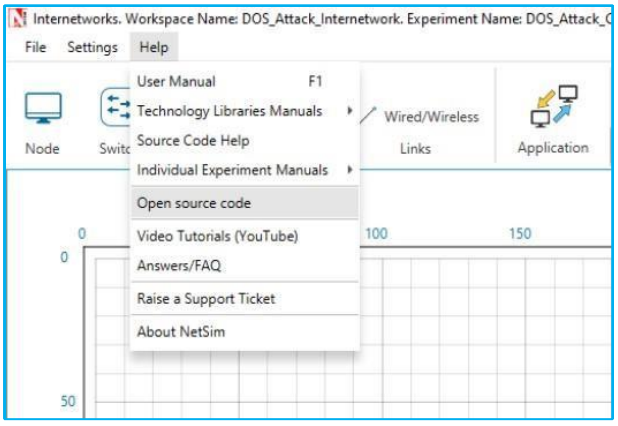
Running Simulations. Case 1: Without Malicious Node

1. Then DOS_Attack_Internetworks comes with a sample configuration that is already saved. To open this example, go to Your work and click on the DOS_Attack_Inw_Case_1 that is present under the list of experiments as shown below:

2. The saved network scenario consisting of 2 Wired Nodes, 1 L2 Switch, 2 router, 1 Access Point and 1 wireless node in the grid environment forming a internetworks Network. Traffic is configured from Wired node to the Wireless node.



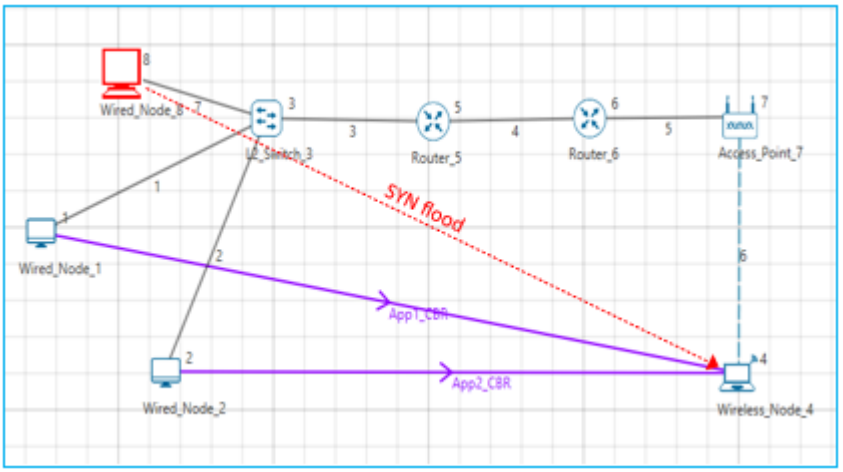
3. Help □ Open Source code



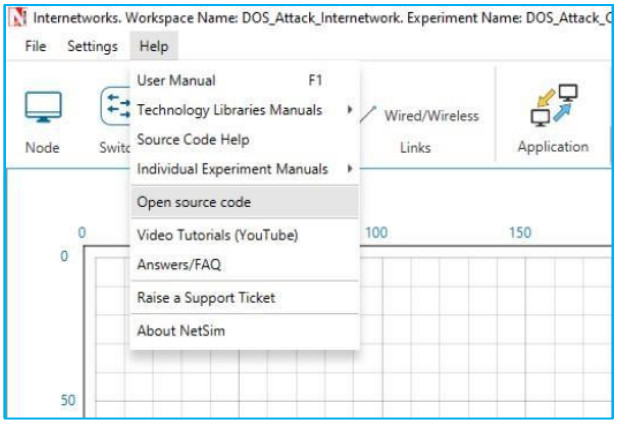
4. In TCP.h set **NUMBEROFMALICIOUSNODE** as 1.
5. In SYN_FLOOD.c set **malicious node** as 0**.**
6. Right click on the solution in the solution explorer and select Rebuild. (Note: first rebuild the TCP project and then rebuild the Ethernet project)
7. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries.
8. Run the simulation for 10 seconds.

Case-2: With one Malicious Node

1. Then DOS_Attack_Internetworks comes with a sample configuration that is already saved. To open this example, go to Your work and click on the DOS_Attack_Inw_Case_2 that is present under the list of experiments as shown below:
2. The saved network scenario consisting of 3 Wired Nodes, 1 L2 Switch, 2 router, 1 Access Point and 1 wireless node in the grid environment forming a internetworks Network. Traffic is configured from Wired node to the Wireless node.



3. Help Open Source code

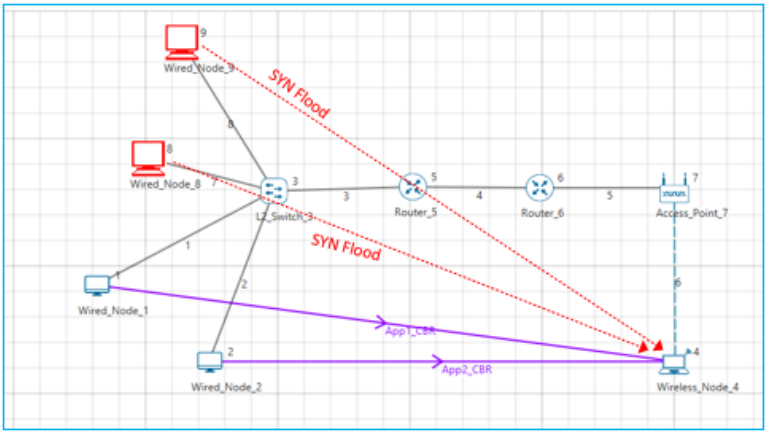


4. In TCP.h set **NUMBEROFMALICIOUSNODE** as 1.
5. In SYN_FLOOD.c set **malicious node** as 8**.**
6. Right click on the solution in the solution explorer and select Rebuild. (Note: first rebuild the TCP project and then rebuild the Ethernet project)
7. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries.
8. Run the simulation for 10 seconds.

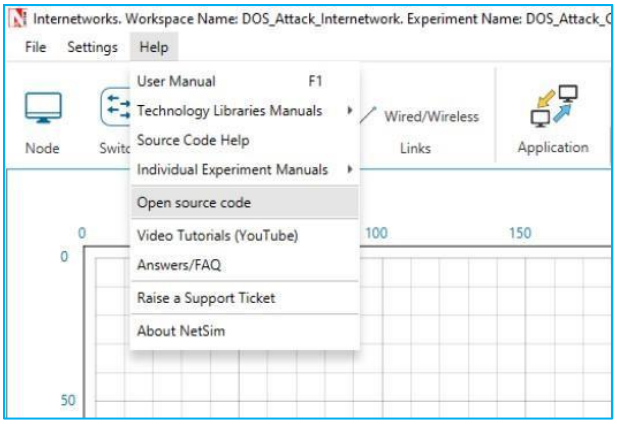
Case-3: With two Malicious Node

1. Then DOS_Attack_Internetworks comes with a sample configuration that is already saved. To open this example, go to your work and click on the DOS_Attack_Inw_Case_3 that is present under the list of experiments.

2. The saved network scenario consisting of 4 Wired Nodes, 1 L2 Switch, 2 router, 1 Access Point and 1 wireless node in the grid environment forming a internetworks Network. Traffic is configured from Wired node to the Wireless node.



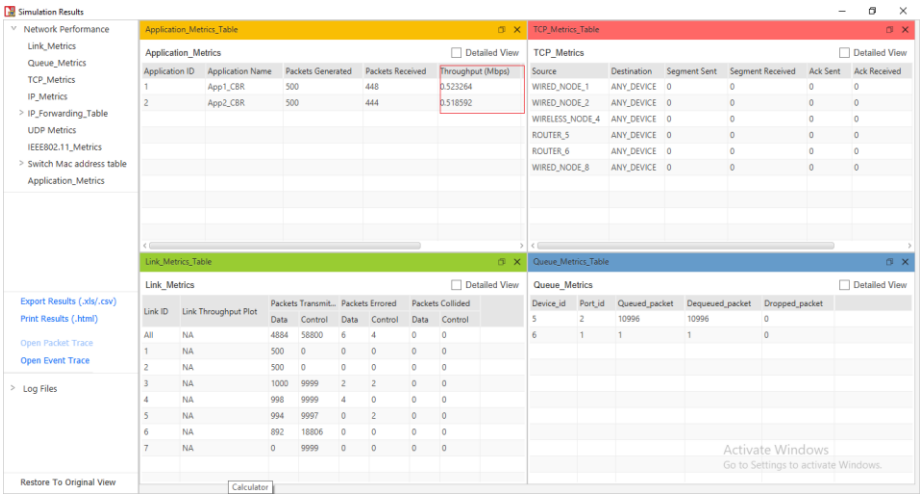
3. Help Open Source code



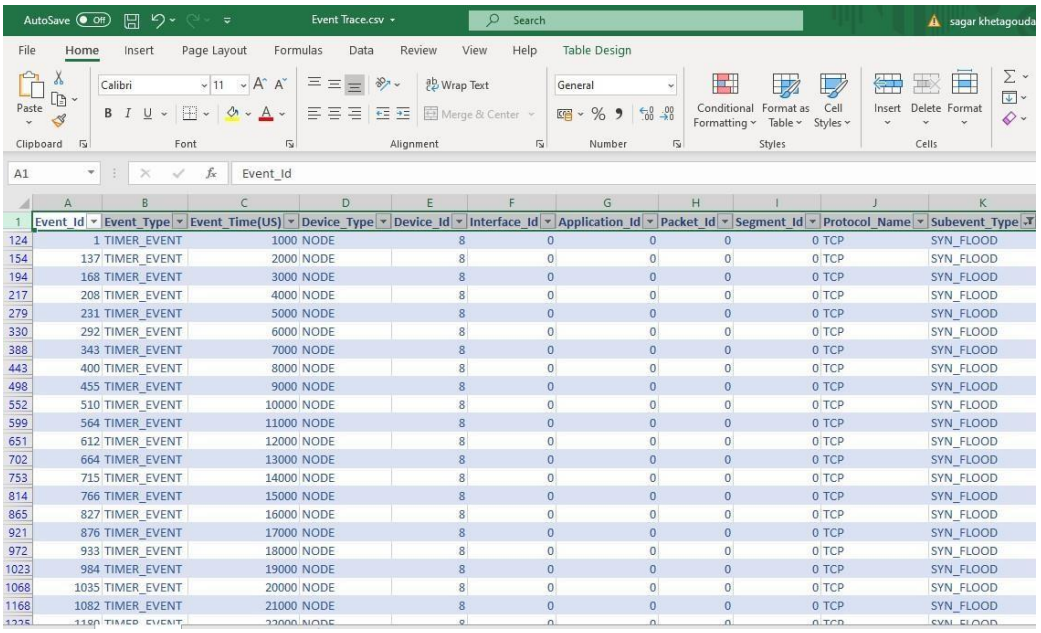
4. In TCP.h set **NUMBEROFMALICIOUSNODE** as 2.
5. In SYN_FLOOD.c set **malicious node** as 8, 9**.**
6. Right click on the solution in the solution explorer and select Rebuild. (Note: first rebuild the TCP project and then rebuild the Ethernet project)
7. Upon successful build modified libTCP.dll and libEthernet.dll file gets automatically updated in the directory containing NetSim binaries.

Results and discussion

After simulation, open metrics window and observe the Throughput.



Go to the result window open Event trace, user can find out the SYN_FLOOD packets via filtering subevent type as SYN_FLOOD.



	A	B	C	D	E	F	G	H	I	J	K
1	Event_Id	Event_Type	Event_Time(US)	Device_Type	Device_Id	Interface_Id	Application_Id	Packet_Id	Segment_Id	Protocol_Name	Subevent_Type
124	1	TIMER_EVENT	1000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
154	137	TIMER_EVENT	2000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
194	168	TIMER_EVENT	3000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
217	208	TIMER_EVENT	4000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
279	231	TIMER_EVENT	5000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
330	292	TIMER_EVENT	6000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
388	343	TIMER_EVENT	7000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
443	400	TIMER_EVENT	8000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
498	455	TIMER_EVENT	9000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
552	510	TIMER_EVENT	10000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
599	564	TIMER_EVENT	11000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
651	612	TIMER_EVENT	12000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
702	664	TIMER_EVENT	13000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
753	715	TIMER_EVENT	14000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
814	766	TIMER_EVENT	15000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
865	827	TIMER_EVENT	16000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
921	876	TIMER_EVENT	17000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
972	933	TIMER_EVENT	18000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
1023	984	TIMER_EVENT	19000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
1068	1035	TIMER_EVENT	20000	NODE	8	0	0	0	0	TCP	SYN_FLOOD
1168	1082	TIMER_EVENT	21000	NODE	8	0	0	0	0	TCP	SYN_FLOOD

Note: Users can also create their own network scenarios in Internetworks and run simulation.

Case 1 shows the results when there is no attack. The two user applications, attain a throughput of about 0.58 Mbps. In the table we see the throughput of for these two applications falling as we increase the number of attack nodes. This is because the server's resources are being used up in handling the SYN-FLOOD packets and the server is unable to sustain packet transmissions for the regular applications. In this example, with a coordinated attack involving 4 systems the throughputs are down 70%.

	Throughput_APP1 (Mbps)	Throughput_APP2 (Mbps)
Case-1: Malicious Node =0	0.5805	0.5782
Case-2: Malicious Node =1	0.5233	0.5186
Case-3: Malicious Node =2	0.2873	0.2862

Table 1: Throughputs seen by the user applications. The first row is the throughput when there is no attack. In other samples show the fall in throughputs as the number of attacker systems are increased

Appendix: NetSim source code modifications

Changes to fn_NetSim_TCP_Trace(), in TCP.c file, within TCP project

```
/* This is used to add the SYN_FLOOD sub-events in Event Trace file */
_declspec (dllexport) char *fn_NetSim_TCP_Trace(int nSubEvent)
{
if (nSubEvent == SYN_FLOOD)
return "SYN_FLOOD";
return (GetStringTCP_Subevent(nSubEvent));
}
```

Changes to fn_NetSim_TCP_HandleTimer(), in TCP.c file, within TCP project

```
/* This is used to call the syn_flood() function periodically */
static int fn_NetSim_TCP_HandleTimer()
{
switch (pstruEventDetails->nSubEventType)
{
case SYN_FLOOD:
syn_flood();
break;
case TCP_RTO_TIMEOUT:
handle_rto_timer();
break;
```

Changes to fn_NetSim_TCP_Init(), in TCP.c file, within TCP project

```
/* This is used to register the first SYN_FLOOD event */
```

```
_declspec (dllexport) int fn_NetSim_TCP_Init(struct stru_NetSim_Network
*NETWORK_Formal,
 NetSim_EVENTDETAILS *pstruEventDetails_Formal,
 char *pszAppPath_Formal,
 char *pszWritePath_Formal,
 int nVersion_Type,
 void **fnPointer)
{
fn_NetSim_TCP_Init_F(NETWORK_Formal,
pstruEventDetails_Formal,
pszAppPath_Formal,
pszWritePath_Formal,
nVersion_Type,
fnPointer);
NetSim_EVENTDETAILS pevent;
memcpy(&pevent, pstruEventDetails, sizeof pevent);
for (int i = 0; i < NETWORK->nDeviceCount; i++)
{
if (is_malicious_node(i + 1))
{
pevent.nDeviceId = i + 1;
pevent.dEventTime += 1000;
pevent.nEventType = TIMER_EVENT;
pevent.nSubEventType = SYN_FLOOD;
pevent.nProtocolId = TX_PROTOCOL_TCP;
fnpAddEvent(&pevent);
}
}
return 0;
}
```

Changes to add_timeout_event() in RTO.c file, within TCP project

```
/* This is used to avoid RTO timeouts for malicious nodes */
void add_timeout_event(PNETSIM_SOCKET s,
 NetSim_PACKET* packet)
```

```
{
NetSim_PACKET* p = fn_NetSim_Packet_CopyPacket(packet);
add_packet_to_queue(&s->tcb->retransmissionQueue, p, pstruEventDetails->dEventTime);
NetSim_EVENTDETAILS pevent;
memcpy(&pevent, pstruEventDetails, sizeof pevent);
pevent.dEventTime += TCP_RTO(s->tcb);
pevent.dPacketSize = packet->pstruTransportData->dPacketSize;
pevent.nEventType = TIMER_EVENT;
pevent.nPacketId = packet->nPacketId;
if (packet->pstruAppData)
{
pevent.nApplicationId = packet->pstruAppData->nApplicationId;
pevent.nSegmentId = packet->pstruAppData->nSegmentId;
}
else
pevent.nSegmentId = 0;
if (!is_malicious_node(pevent.nDeviceId))
{
pevent.nProtocolId = TX_PROTOCOL_TCP;
pevent.pPacket = fn_NetSim_Packet_CopyPacket(p);
pevent.szOtherDetails = NULL;
pevent.nSubEventType = TCP_RTO_TIMEOUT;
fnpAddEvent(&pevent);
print_tcp_log("Adding RTO Timer at %0.1lf", pevent.dEventTime);
}
}
```

Changes to TCP.h file, within TCP project

```
/* This is used to define the number of malicious nodes */
#pragma comment (lib,"NetworkStack.lib")
_declspec(dllexport) target_node;
//USEFUL MACRO
#define isTCPConfigured(d) (DEVICE_TRXLayer(d) && DEVICE_TRXLayer(d)->isTCP)
#define isTCPControl(p) (p->nControlDataType/100 == TX_PROTOCOL_TCP)
//Constant
#define TCP_DupThresh 3
#define NUMBEROFMALICIOUSNODE 2
```

Addition of SYN_flood.c file, within TCP project

```
/* This is used to define the malicious node ID's and the target node ID */
/* This has functions defined for SYN flood attack*/
#include "main.h"
#include "TCP.h"
#include "List.h"
#include "TCP_Header.h"
#include "TCP_Enum.h"
int malicious_node[NUMBEROFMALICIOUSNODE] = {2,6};
static void send_syn_packet(PNETSIM_SOCKET s);
//static PNETSIM_SOCKET socket_creation();
int target_node = 4;
PNETSIM_SOCKET get_Remotesocket(NETSIM_ID d, PSOCKETADDRESS addr);
static PSOCKETADDRESS sockAddr = NULL;
int is_malicious_node(NETSIM_ID devid){}
void syn_flood(){}
static void send_syn_packet(PNETSIM_SOCKET s){}
int socket_creation(){}
```

Changes to TCP_Enum.h file, within TCP project

```
/* This is used to a new SYN_FLOOD subevent in TCP_Subevent */
#include "EnumString.h"
BEGIN_ENUM(TCP_Subevent)
{
DECL_ENUM_ELEMENT_WITH_VAL(TCP_RTO_TIMEOUT, TX_PROTOCOL_TCP *
100),
DECL_ENUM_ELEMENT(TCP_TIME_WAIT_TIMEOUT),
DECL_ENUM_ELEMENT(SYN_FLOOD),
}
```

Changes to Ethernet.h file, within ETHERNET project

```
/* This is used to define processing time for syn_flood packets */
#ifndef _NETSIM_ETHERNET_H_
#define _NETSIM_ETHERNET_H_
#ifdef __cplusplus
extern "C" {
#endif
#pragma comment(lib,"NetworkStack.lib")
#pragma comment(lib,"Metrics.lib")
#pragma comment (lib,"libTCP.lib")
#define isETHConfigured(d,i) (DEVICE_MACLAYER(d,i)->nMacProtocolId ==
```

MAC_PROTOCOL_IEEE802_3)

//Global variable

PNETSIM_MACADDRESS multicastSPTMAC;

#define ETH_IFG 0.960 //Micro sec

#define Processing_TIME 1000

Changes to fn_NetSim_Ethernet_HandlePhyOut() in Ethernet_Phy.c file, within ETHERNET project

/* This is used to add processing delay for TCP SYN packets */

/* This is used to add processing delay for TCP SYN packets */

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;

}

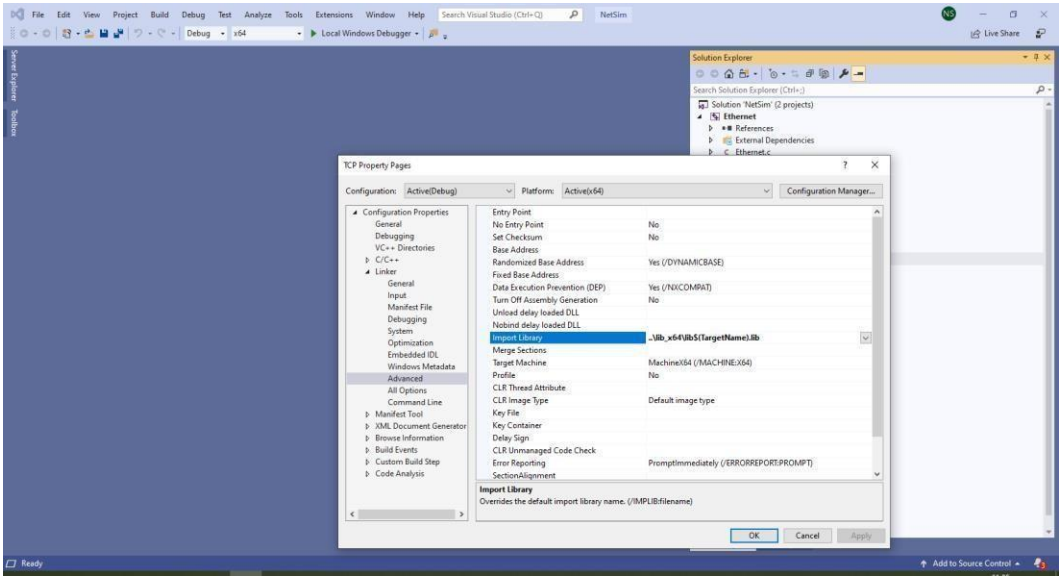
TCP Project Properties:

• Right click on TCP project and select Properties.

• In Linker section go to Advanced

• The import library value has been updated for 64-bit source code settings.

o 64-bit as ..\lib_**x64\lib$(TargetName).lib**


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search Visual Studio (Ctrl+Q)
NetSim
Debug
x64
Local Windows Debugger
Live Share
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'NetSim' (2 projects)
Ethernet
References
External Dependencies
Ethernet.c
TCP Property Pages
Configuration: Active(Debug)
Platform: Active(x64)
Configuration Manager...
Configuration Properties
General
Debugging
VC++ Directories
C/C++
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Windows Metadata
Advanced
All Options
Command Line
Manifest Tool
XML Document Generator
Browse Information
Build Events
Custom Build Step
Code Analysis
Entry Point
No Entry Point No
Set Checksum No
Base Address
Randomized Base Address Yes (/DYNAMICBASE)
Fixed Base Address
Data Execution Prevention (DEP) Yes (/NXCOMPAT)
Turn Off Assembly Generation No
Unload delay loaded DLL
Nobind delay loaded DLL
Import Library ..\lib_x64\lib$(TargetName).lib
Merge Sections
Target Machine MachineX64 (/MACHINE:X64)
Profile No
CLR Thread Attribute
CLR Image Type Default image type
Key File
Key Container
Delay Sign
CLR Unmanaged Code Check
Error Reporting PromptImmediately (/ERRORREPORT:PROMPT)
SectionAlignment
Import Library
Overrides the default import library name. (/IMPLIB:filename)
OK
Cancel
Apply
Ready
Add to Source Control