Simulating DoS Attack (Internetworks)

Software: NetSim Standard v13.0 (32/64 bit), Visual Studio 2019

Project Code Download Link:
https://github.com/NetSim-TETCOS/DOS_Attack_in_Internetworks_v13.0/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 act of overwhelming a victim computer's resources thereby making it impossible to service intended users. Clients are denied service from the victim computer. A successful DoS attack starts consuming system resources (memory and compute) leading to a slowdown and eventually a shutdown. When multiple attackers coordinate a DoS attack, it is known as a DDoS (Distributed Denial of Service) attack. The standard types of Dos attacks are:

- SYN Flood
- UDP Flood
- ICMP Flood
- HTTP GET Flood

The SYN Flood Attack

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

- Client requests a connection by sending a SYN (synchronize) message to the server.
- Server acknowledges the request by sending SYN-ACK back to the client.
- 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 attack occurs when an attacker sends a succession of TCP Synchronize (SYN) requests to the target; the SYN request opens network communication between a prospective client (the attacker) and the target server. When the server receives a SYN request, it responds with a SYN-ACK and holds the communication open while it waits for the client to send an ACK.

In a successful SYN-Flood attack the final client ACK 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 resources and paralyzes the machine.

At the malicious (attacker) node

In NetSim, a DOS attack is parameterized by specifying the SYN-FLOOD packet inter-arrival time, which is the time between successive packets. The inter-packet arrival time is the reciprocal of the packet rate.

This project code creates a new timer event called SYN_FLOOD in TCP for sending TCP_SYN packets. This event is called every inter-arrival time. By default, this value is set to 1 ms or 1000 μs (we use μs since this is the unit of time in NetSim). The attacker thus creates and sends a TCP_SYN packet for every 1000 μs. Each SYN request opens one TCP connection request between the attacker and the target.

At the target (victim) node

When the target receives a SYN request, it responds with a SYN-ACK and holds the connection open. It waits for the client (or attacker in this case) to complete the 3-way handshake with an ACK, in response to its SYN-ACK.

Since NetSim is a packet level network simulator it assumes infinite compute and memory capability at the nodes. To overcome this limitation, an abstraction is required to model the impact of each SYN request. In this project, we use “processing time” to account for resource consumption. The idea is that each SYN-ACK, which is sent in response to the SYN, would take some time to be created, initialized, and transmitted into the networks. Thus, time is used in place of memory or compute resources.

The default value of processing time 2000 μs; it is user editable. This implies that when the attacked node responds with a SYN_ACK packet, a processing time of 2000 μs is added[1]. It is worth noting that SYN packets are created every 1000 μs whereas each SYN-ACK takes a 2000 μs processing time. During this processing time, the server is unable to handle all other data communication; other applications start queuing. With an increase in the number of SYN packets, the server starts progressively slowing down. At some threshold the data traffic connections get timed out and the server cannot handle any further communication.

C functions for the SYN_FLOOD attack

To implement this project in NetSim, we create the 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 1000 μs)

The TCP Log file

- Users 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 large file and hence is disabled by default in NetSim.
- Go to TCP.c inside the TCP project and change the function bool isTCPlog() to return true instead of false. This enables logging.

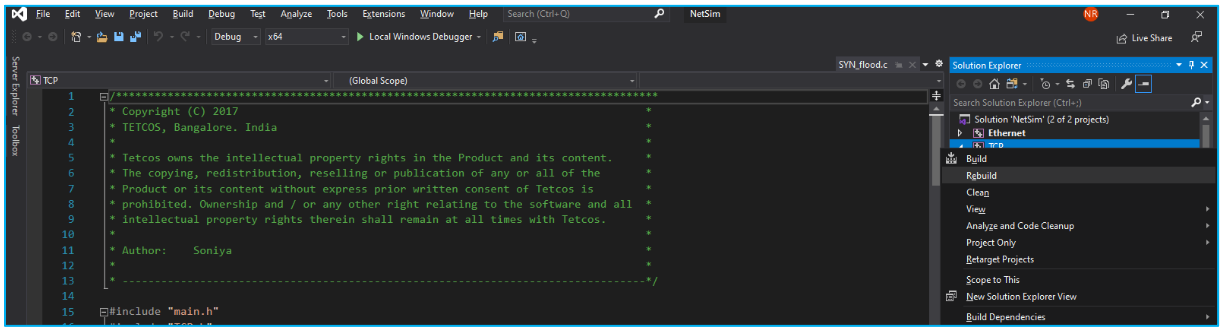
Steps to simulate the attack

1. Open the Source codes in Visual Studio by going to Your work-> Workspace Options and Clicking on Open code button in NetSim Home Screen window.

2. In Visual Studio, under the **TCP** project in the solution explorer, a **SYN_FLOOD.c** file is added as part of this project.

[1] This processing time is added in the Ethernet PHY-OUT event in NetSim.

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).



```
/***********************************************************************************
* Copyright (C) 2017                                                              *
* TETCOS, Bangalore. India                                                        *
*                                                                                 *
* Tetcos owns the intellectual property rights in the Product and its content.    *
* The copying, redistribution, reselling or publication of any or all of the      *
* Product or its content without express prior written consent of Tetcos is       *
* prohibited. Ownership and / or any other right relating to the software and all *
* intellectual property rights therein shall remain at all times with Tetcos.     *
*                                                                                 *
* Author:    Soniya                                                               *
*                                                                                 *
* ---------------------------------------------------------------------------------*/

#include "main.h"
```

Figure 1: Screen shot of NetSim project source code in Visual Studio

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 an attacker (malicious nodes)

1. The DOS_Attack_Internetworks 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 DOS_Attack_Case_1 from the list of experiments.

2. The saved network scenario consists of
    a. 2 Wired Nodes
    b. 1 L2 Switch
    c. 2 Routers
    d. 1 Access Point and
    e. 1 wireless node

forming a network. Regular application traffic is configured from the Wired nodes to the Wireless node. These applications are named as User-1-DL and User-2-DL. The scenario screen shot is shown below.

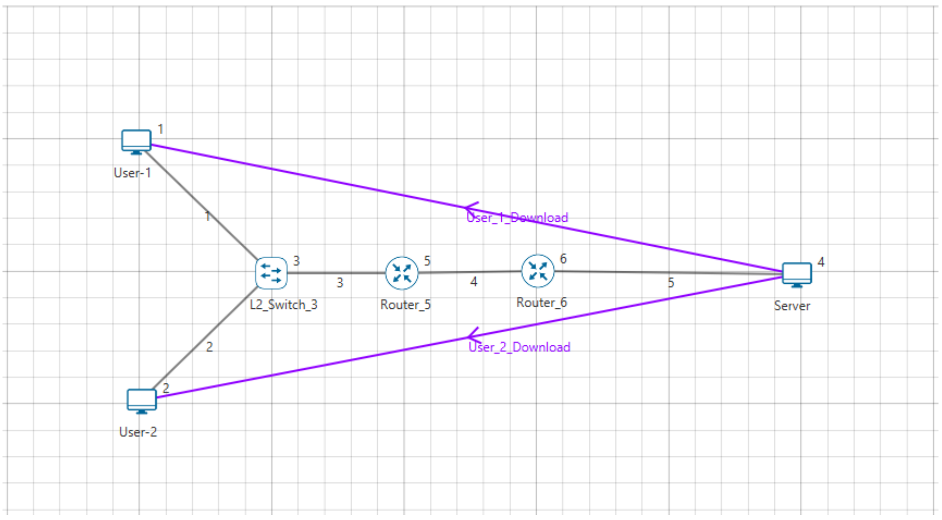


Figure 2: Model of regular client nodes communicating with the server in NetSim

3. Help > Open-source code

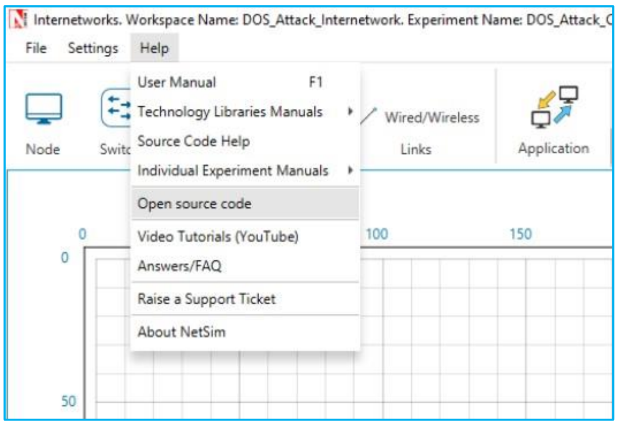

Internetworks. Workspace Name: DOS_Attack_Internetwork. Experiment Name: DOS_Attack_C
File
Settings
Help
User Manual F1
Technology Libraries Manuals
Source Code Help
Individual Experiment Manuals
Open source code
Video Tutorials (YouTube)
Answers/FAQ
Raise a Support Ticket
About NetSim
Node
Wired/Wireless Links
Application

Figure 3: Open-source code in one click

4. In TCP.h set **NUMBEROFMALICIOUSNODE** as 1.

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

Case 2: With one Malicious Node

1. The 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_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.

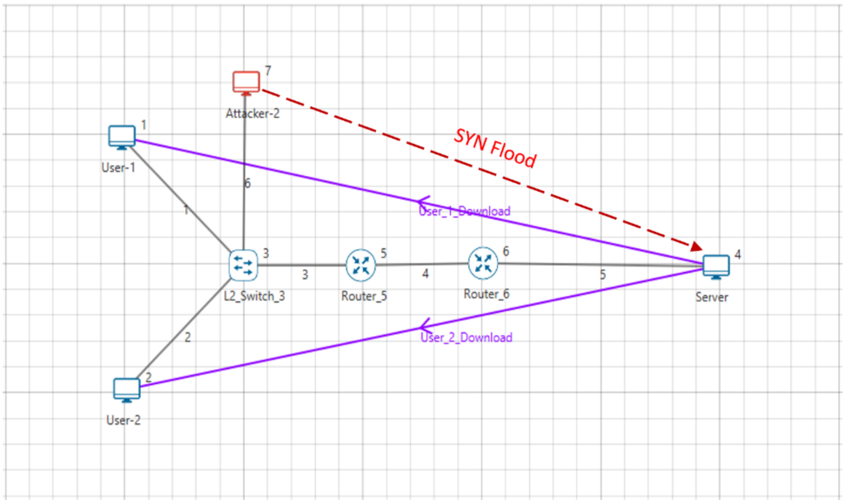


Figure 4: A malicious node initiates a SYN-FLOOD attack

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 multiple Malicious Nodes

1. The 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_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.

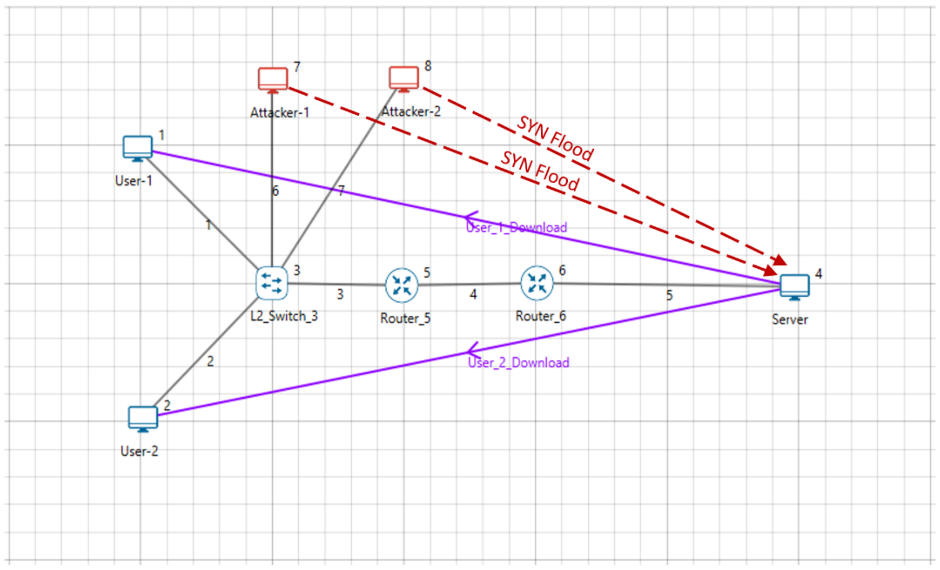
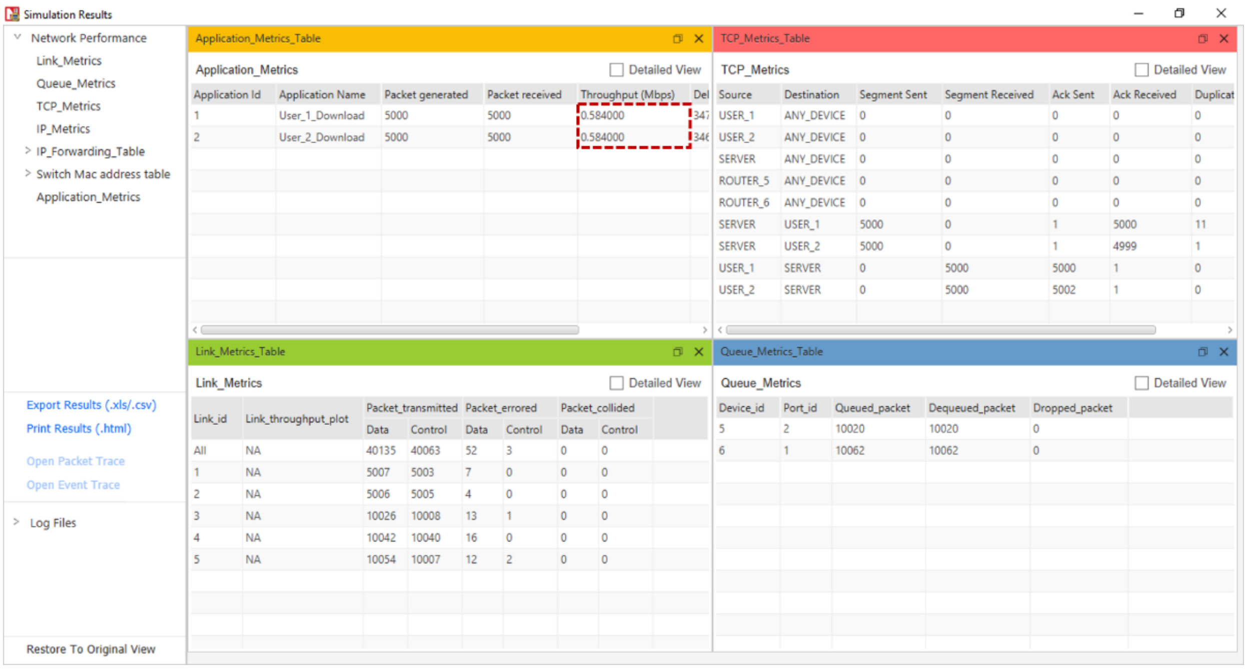


Figure 5: Now two malicious nodes are involved the SYN-FLOOD attack

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.
8. Run the simulation for 10 seconds.
9. Repeat the steps for 3 and 4 attacker nodes.

Results and discussion

After simulation open metrics window and observe the throughput.


Simulation Results

Network Performance
- Link_Metrics
- Queue_Metrics
- TCP_Metrics
- IP_Metrics
- IP_Forwarding_Table
- Switch Mac address table
- Application_Metrics

Export Results (.xls/.csv)
Print Results (.html)
Open Packet Trace
Open Event Trace
Log Files
Restore To Original View

Application_Metrics_Table

Application_Metrics

Application Id	Application Name	Packet generated	Packet received	Throughput (Mbps)	Del
1	User_1_Download	5000	5000	0.584000	347
2	User_2_Download	5000	5000	0.584000	346

TCP_Metrics_Table

TCP_Metrics

Source	Destination	Segment Sent	Segment Received	Ack Sent	Ack Received	Duplicat
USER_1	ANY_DEVICE	0	0	0	0	0
USER_2	ANY_DEVICE	0	0	0	0	0
SERVER	ANY_DEVICE	0	0	0	0	0
ROUTER_5	ANY_DEVICE	0	0	0	0	0
ROUTER_6	ANY_DEVICE	0	0	0	0	0
SERVER	USER_1	5000	0	1	5000	11
SERVER	USER_2	5000	0	1	4999	1
USER_1	SERVER	0	5000	5000	1	0
USER_2	SERVER	0	5000	5002	1	0

Link_Metrics_Table

Link_Metrics

Link_id	Link_throughput_plot	Packet_transmitted		Packet_errored		Packet_collided	
		Data	Control	Data	Control	Data	Control
All	NA	40135	40063	52	3	0	0
1	NA	5007	5003	7	0	0	0
2	NA	5006	5005	4	0	0	0
3	NA	10026	10008	13	1	0	0
4	NA	10042	10040	16	0	0	0
5	NA	10054	10007	12	2	0	0

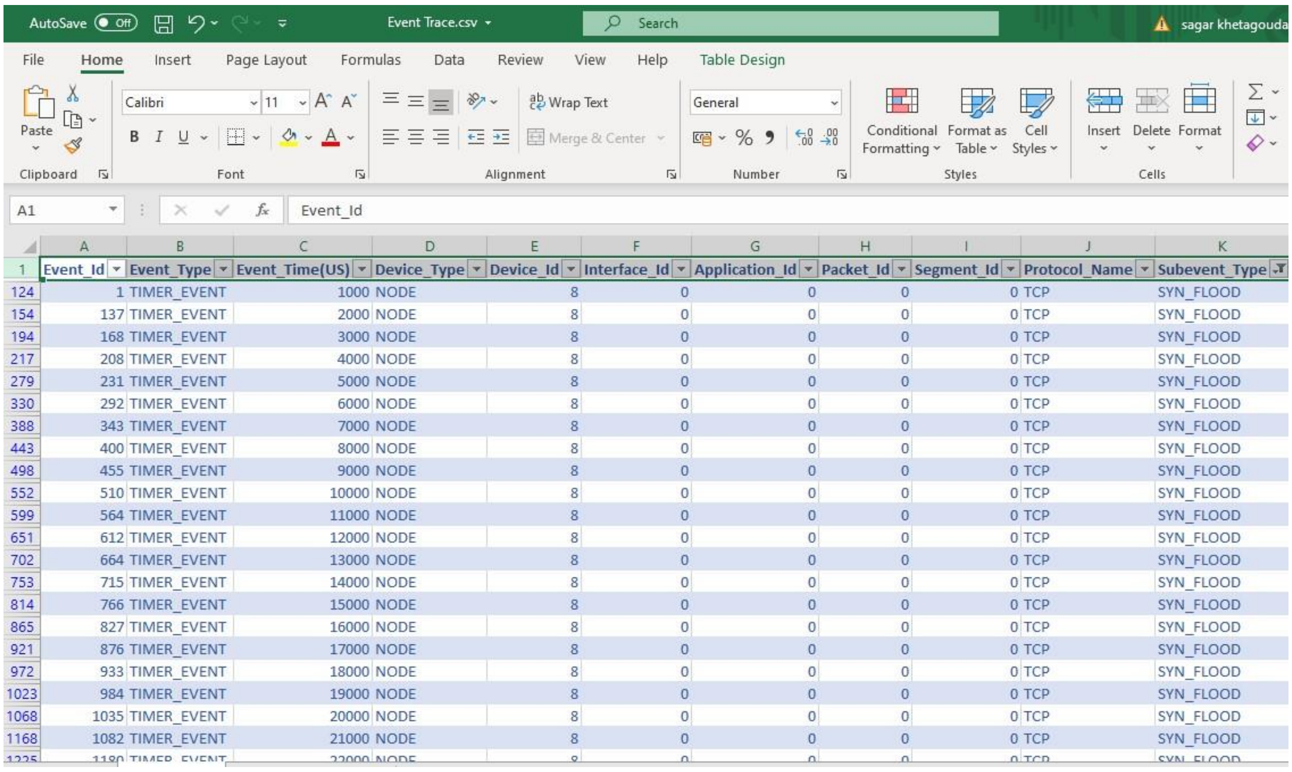
Queue_Metrics_Table

Queue_Metrics

Device_id	Port_id	Queued_packet	Dequeued_packet	Dropped_packet
5	2	10020	10020	0
6	1	10062	10062	0

Fig 6: NetSim results dashboard with throughput highlighted

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



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
1225	1180	TIMER_EVENT	22000	NODE	8	0	0	0	0	TCP	SYN_FLOOD

Figure 7: NetSim Event trace with filtered applied to SUB-EVENT-TYPE column. It shows all the SYN_FLOOD Packets and one can notice the 1000 µs inter-packet arrival times in Column C

Case 1 shows the results when there is no attack. The two user applications, User-1-DL and User-2-DL, 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 co-ordinated attack involving 4 systems the throughputs are down 70%.

Number of attack systems	**User-1-DL (Mbps)**	**User-2-DL (Mbps)**
N/A	0.5804	0.5804
1	0.5233	0.5186
2	0.2873	0.2862
3	0.1973	0.1985
4	0.1518	0.1518

Table 1: Throughputs seen by the user applications. The first row is the throughput when there is no attack. The 2nd through 5th rows shows the fall in throughputs as the number of attacker systems are increased.

Users can similarly run DOS attack simulations on their own networks and analyse its impact on throughput and latency.

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 4
```

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] = {7,8,9,10};
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
```

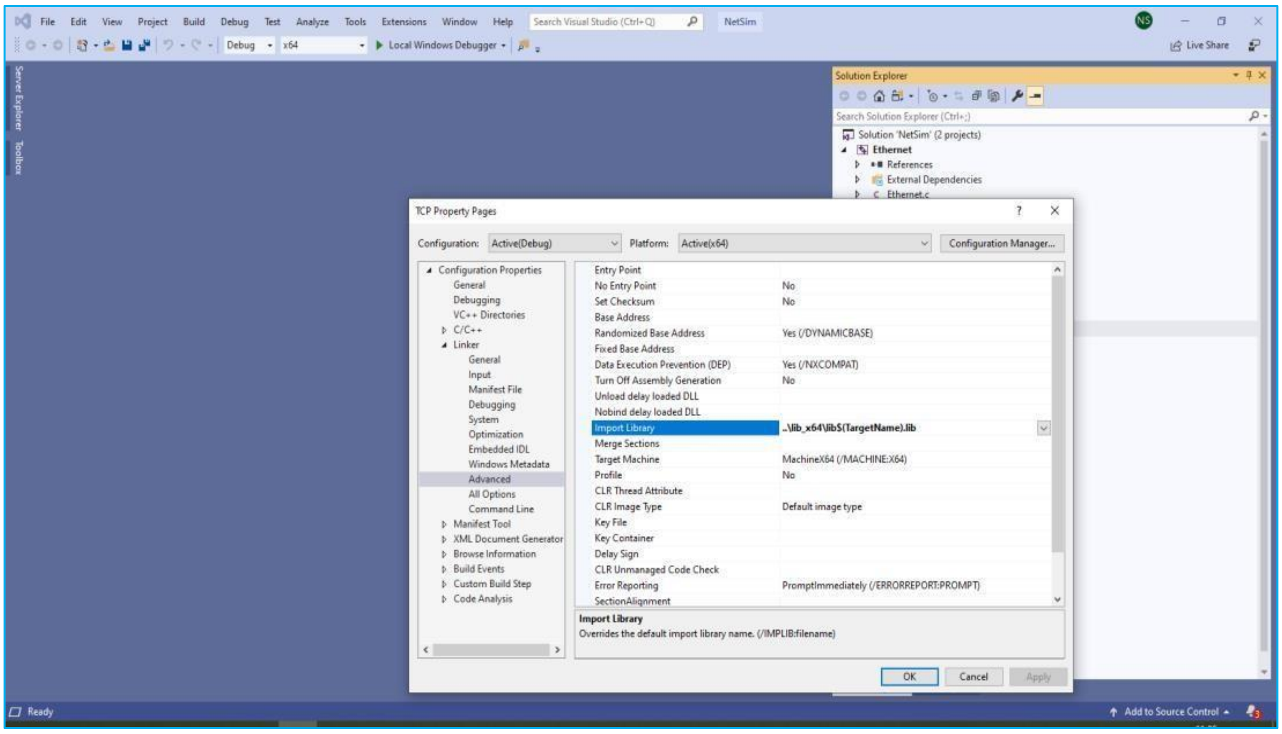
Changes to fn_NetSim_Ethernet_HandlePhyOut() in Ethernet_Phy.c file, within ETHERNET project

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

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;
} function present in Ethernet_Phy.c file inside Ether
```

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 32-bit and 64-bit source code settings.
    - 32-bit as ***..\lib\lib$(TargetName).lib***
    - 64-bit as ***..\lib_x64\lib$(TargetName).lib***


TCP Property Pages

Configuration: Active(Debug)   Platform: Active(x64)

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)

Figure 8: Visual Studio project settings