Sink Hole Attack using RPL in IOT

Software Recommended: NetSim Standard v13.0 (32-bit/ 64-bit), Visual Studio 2017/2019

Follow the instructions specified in the following link to clone/download the project folder from GitHub using Visual Studio:

https://tetcos.freshdesk.com/support/solutions/articles/14000099351-how-to-clone-netsim-file-exchange-project-repositories-from-github-

Other tools such as GitHub Desktop, SVN Client, Sourcetree, Git from the command line, or any client you like to clone the Git repository.

Note: It is recommended not to download the project as an archive (compressed zip) to avoid incompatibility while importing workspaces into NetSim.

Secure URL for the GitHub repository:

In sinkhole Attack, a compromised node or malicious node advertises fake rank information to form the fake routes. After receiving the message packet, it drop the packet information. Sinkhole attacks affect the performance of IoT networks protocols such as RPL protocol.

Implementation in RPL (for 1 sink)

- In RPL the transmitter broadcasts the DIO during DODAG formation.
- The receiver on receiving the DIO from the transmitter updates its parent list, sibling list, rank and sends a DAO message with route information.
- Malicious node upon receiving the DIO message it does not update the rank instead it always advertises a fake rank.
- The other node on listening to the malicious node DIO message the update their rank according to the fake rank.
- After the formation of DODAG, if the node that is transmitting the packet has malicious node as the preferred parent, transmits the packet to it but the malicious node instead of transmitting the packet to its parent, it simply drops the packet resulting in zero throughput.

A file Malicious.c is added to the RPL project.
The file contains the following functions

1. **fn_NetSim_RPL_MaliciousNode( )**
   This function is used to identify whether a current device is malicious or not in-order to establish malicious behaviour.
2. **fn_NetSim_RPL_MaliciousRank( )**
   This function is used to give a fake rank to the malicious node.
3. **rpl_drop_msg( )**
   This function is used to drop the packet by the malicious node if it enters into its network layer.

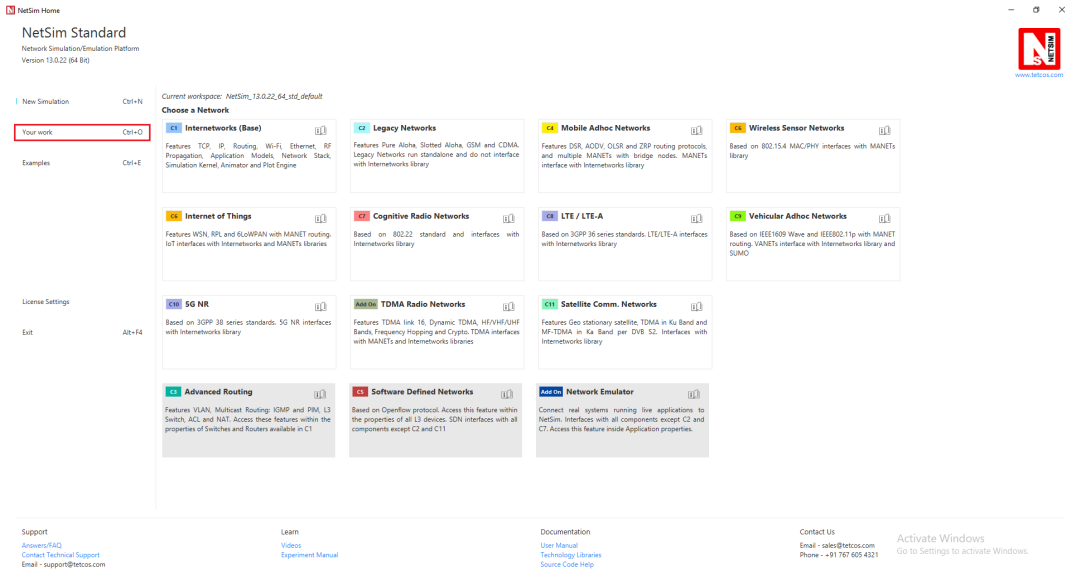
Sink Hole attack – The malicious node advertises the fake rank.
fn_NetSim_RPL_MaliciousRank( ) is the sink hole attack function.

Black Hole attack – The malicious node drops the packet.
rpl_drp_msg() is the black hole attack function

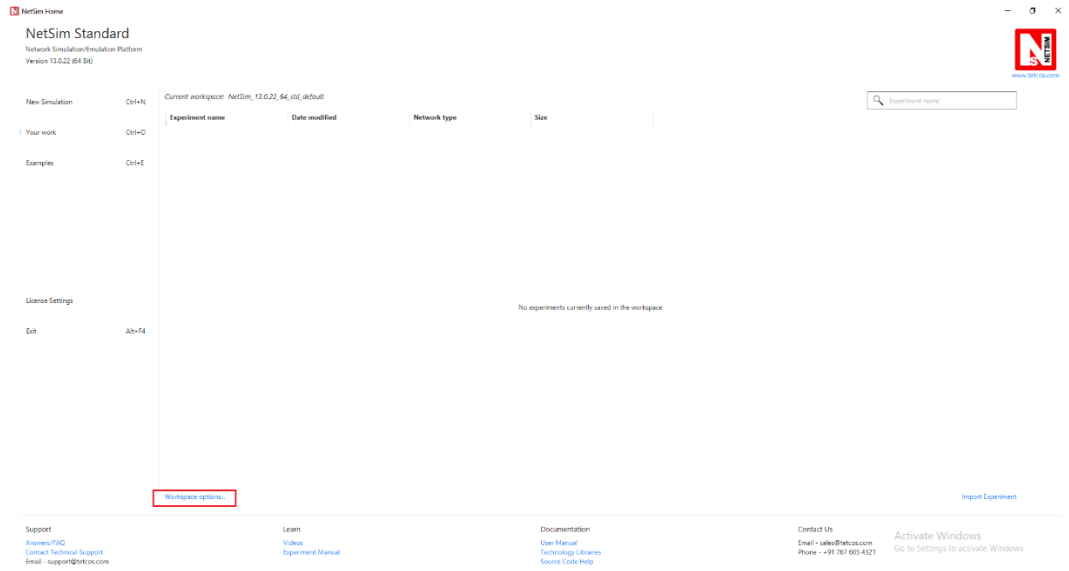
You can set any device as malicious and you can have more than one malicious node in a scenario. Device id's of malicious nodes can be set inside the fn_NetSim_RPL_MaliciousNode() function.

Steps:

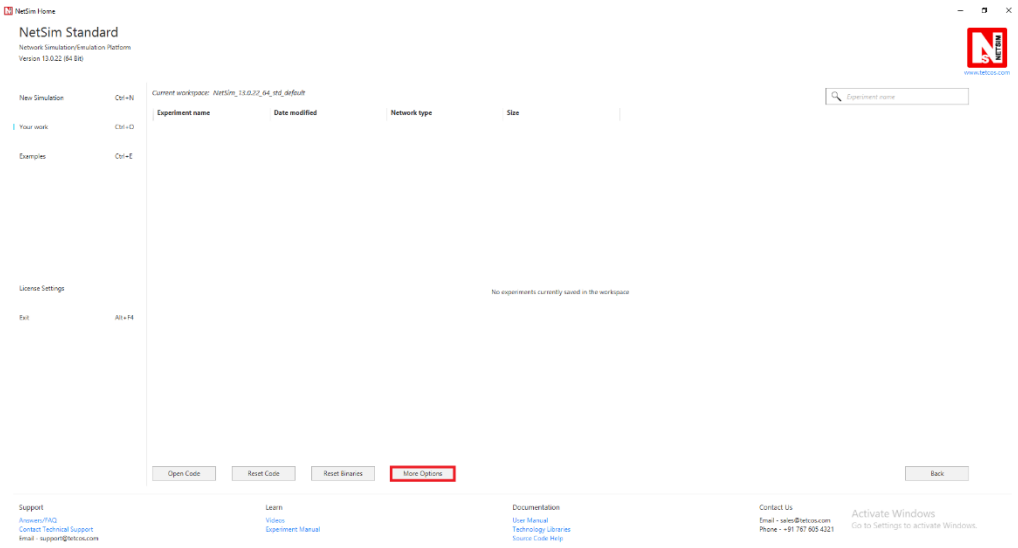
1. After you unzip the downloaded project folder, Open NetSim Home Page click on **Your work** option,



2. Click on **Workspace options**

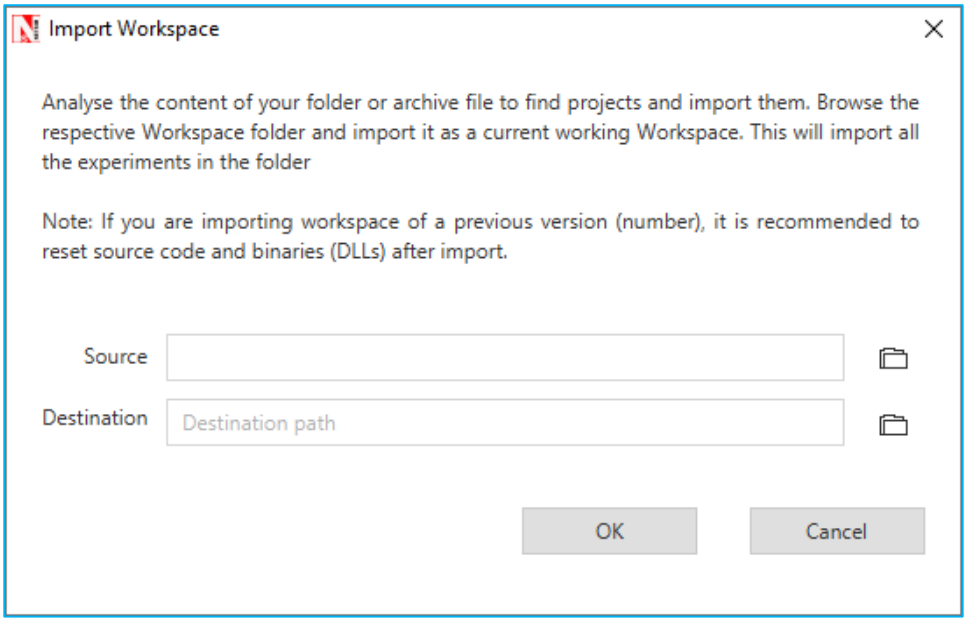


3. Click on **More Options,**

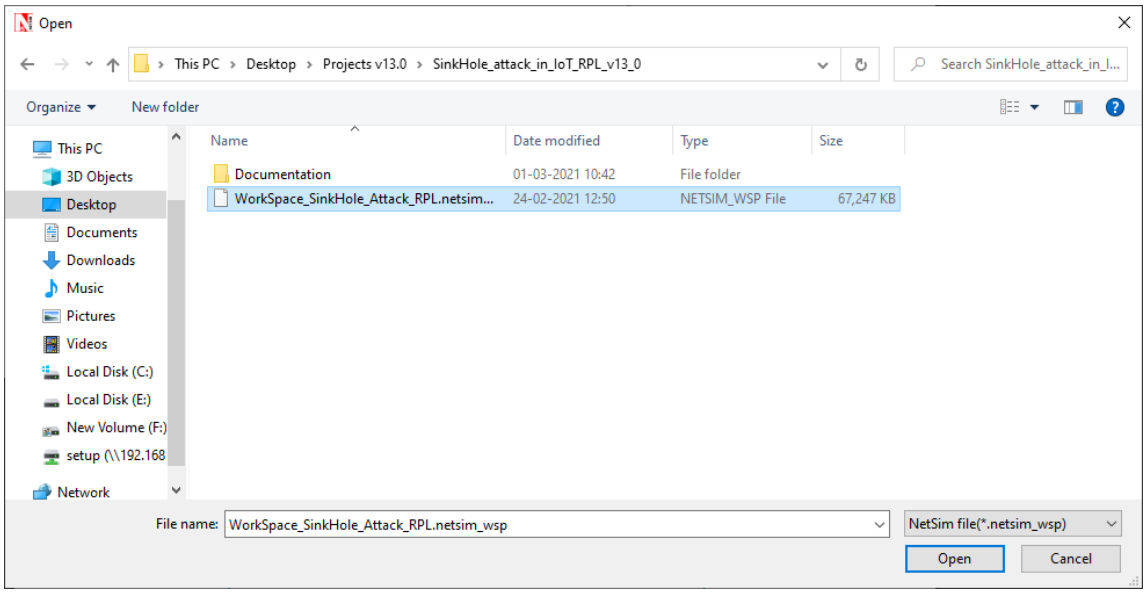


4. This will display a window were users need to give the source file (exported workspace file) and the Destination, the path where the workspace is to be imported to and then click on ok.

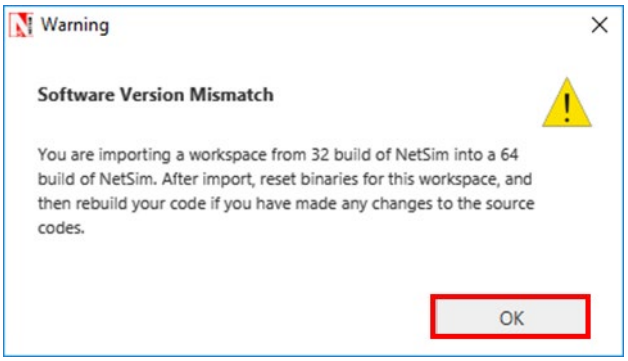
   Note: Only exported workspaces with ".netsim_wsp" extension can be imported



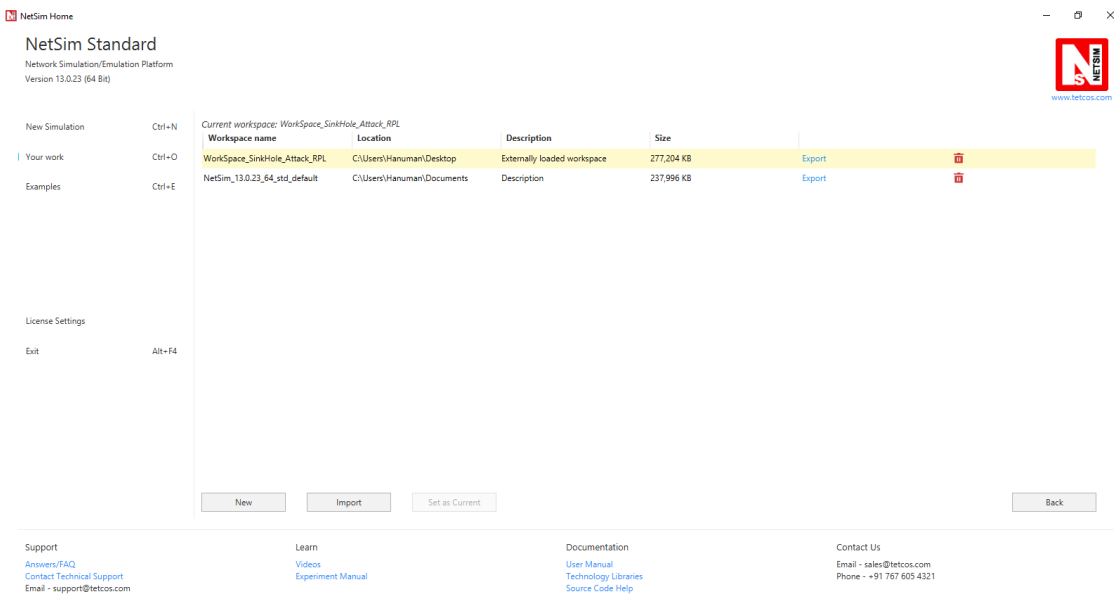
5. Browse to the WorkSpace_SinkHole_Attack_RPL.netsim_wsp folder and click on select folder as shown below:



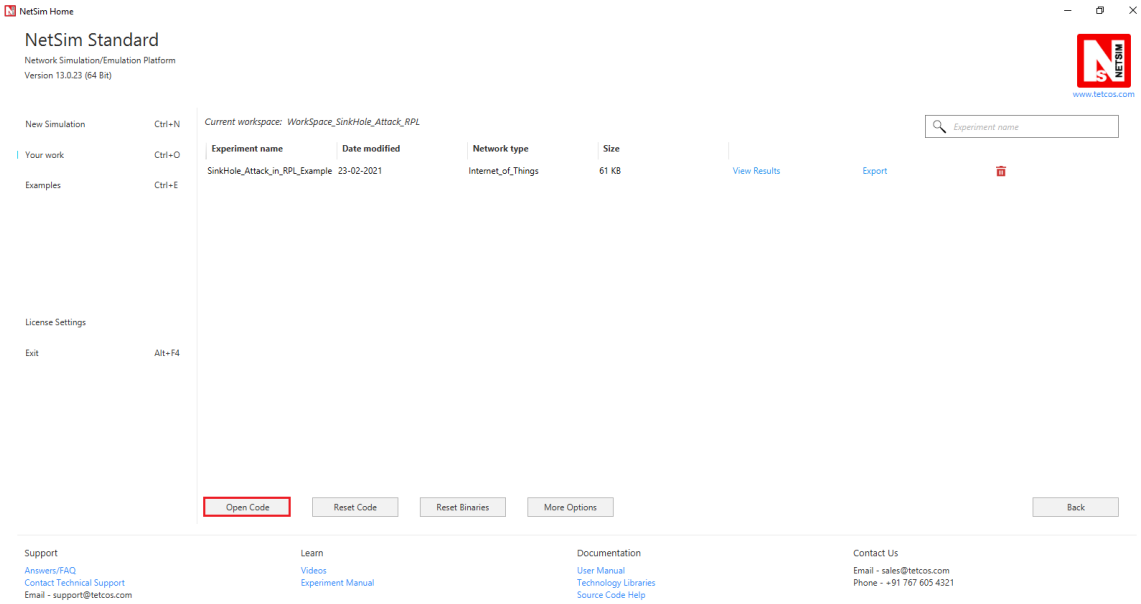
6. After this click on OK button in the Import Workspace window.
7. While importing the workspace, if the following warning message indicating Software Version Mismatch is displayed, you can ignore it and proceed.



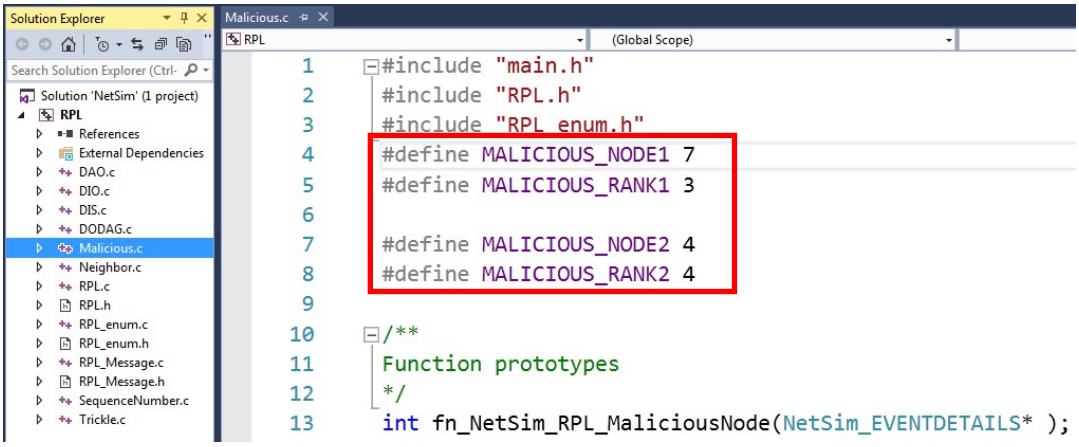
8. The Imported workspace will be set as the current workspace automatically. To see the imported workspace, click on Your work->Workspace Options->More Options as shown below:



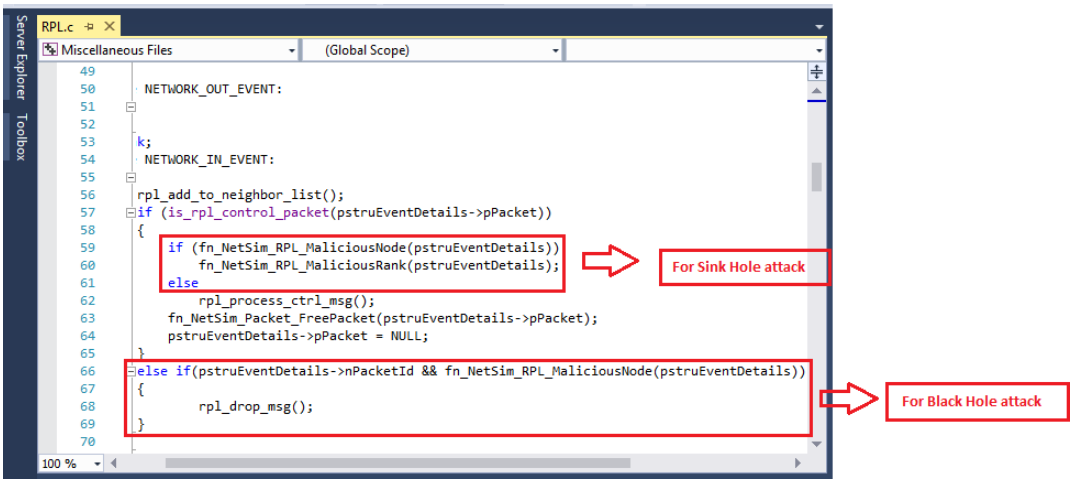
9. Open the Source codes in Visual Studio by going to **Your work-> Workspace Options** and Clicking on Open code button as shown below:



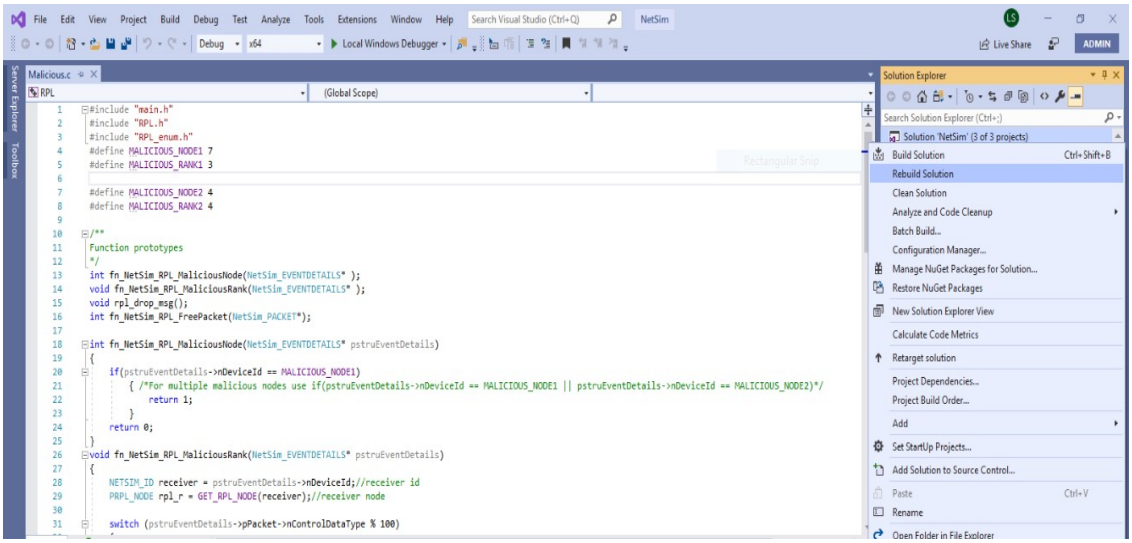
10. Set malicious node id and the fake Rank.



11. Add the code that is highlighted in RPL.c file



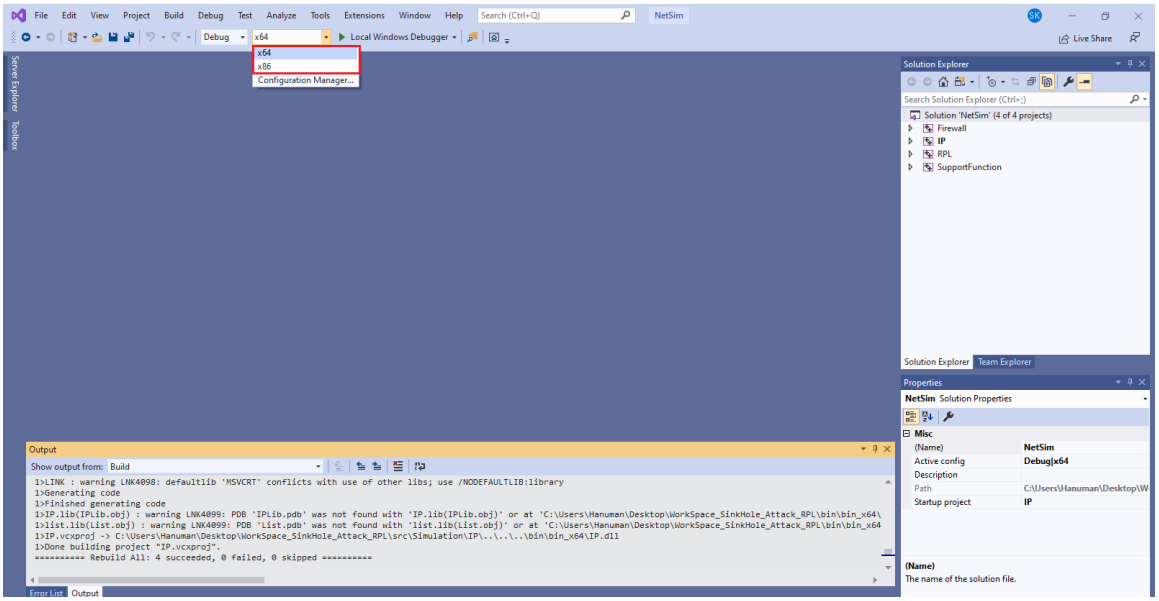
12. Now right click on Solution explorer and select Rebuild.



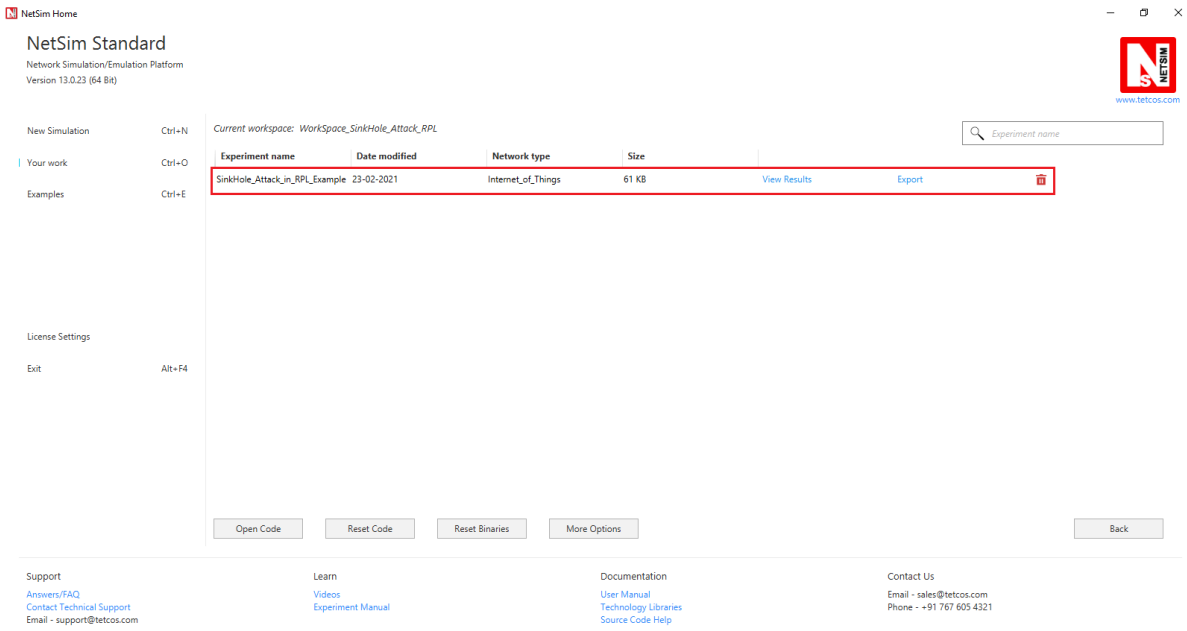
13. Upon rebuilding, **libRPL.dll, libIP.dll,SupportFunction.dll** and **Firewall.dll** will automatically get replaced in the respective bin folders of the current workspace

Note:

1. 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:



14. Go to NetSim home page, click on **Your work** ,Click on SinkHole_Attack_in_RPL_ Example.

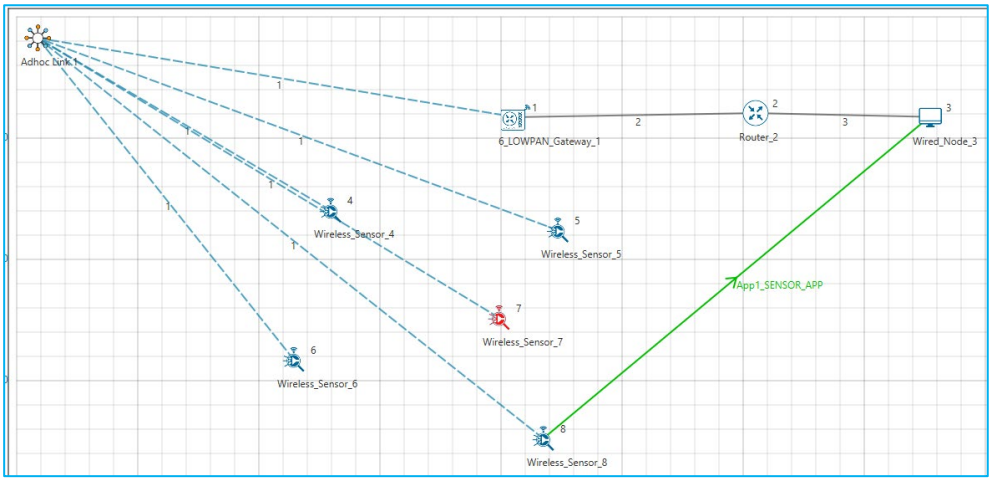


15. Run the simulation for 100 seconds.

Settings that were done to create the network scenario for SinkHole Attack:

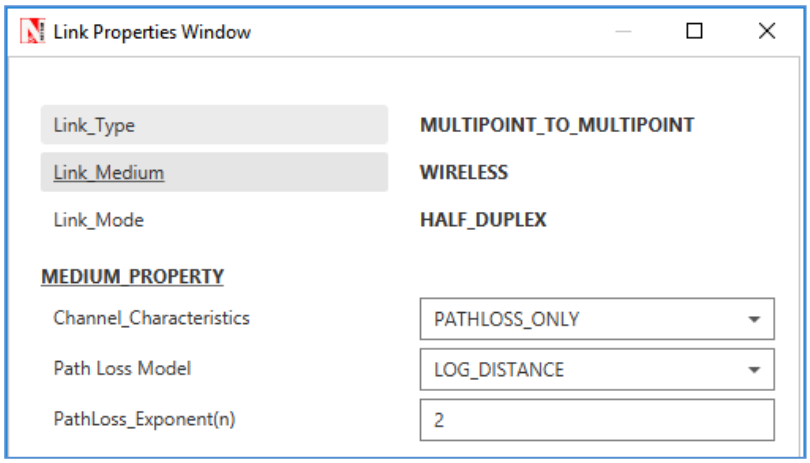
1. Create a network scenario in **IoT (Internet of Things) with UDP running in the Transport Layer and RPL in Network Layer.**

2. For example, you can create a scenario as shown in the following screenshot:



Environment Properties:

- Right click on the Adhoc link icon and select Properties.
- Select the Channel Characteristics and set the parameters accordingly.

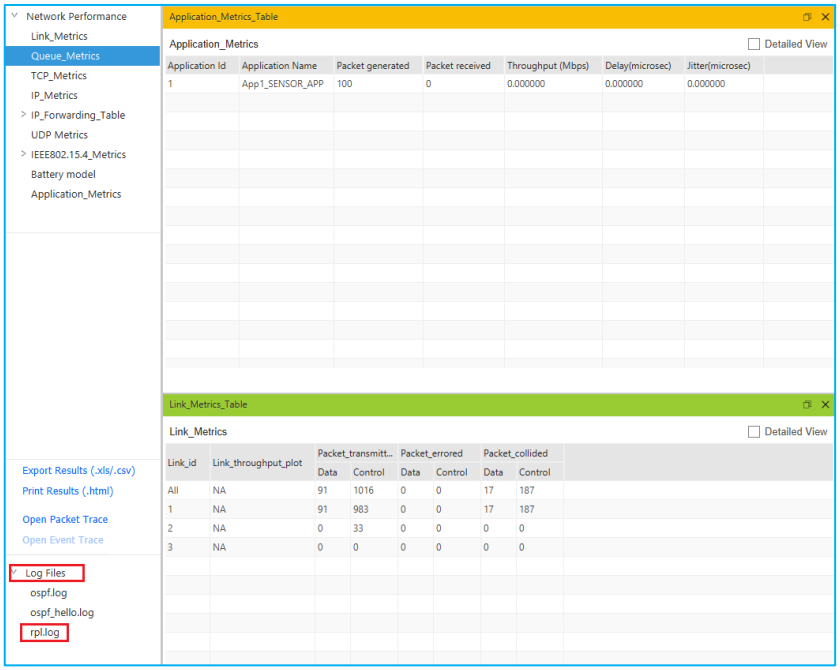


Output

Open **rpllog.txt** file from simulation results window, then you will find the information about DODAG formation.

For every DODAG, 6LoWPAN Gateway is the root of the DODAG

- Root is 1 with rank = 1 (Since the Node Id_1 is 6LoWPAN Gateway)
- Wireless_Sensor_Node_7(Malicious Node)



Packet is **transmitted** by **node 8(Sensor_8)** is **received** by **node 7(Sensor_7)** since the node **7 is malicious** node it drops the packet. So the **Throughput** in this scenario is **0**.

Open **Packet trace** file from simulation results window and filter the **Control Packet Type/App Name** to **App1_ Sensor _App**.

Check the data packets flow, the **Transmitter_Id and receiver_Id** column. Since the node 7 is malicious node it drops the packet without forwarding it further.

1	PACKET_ID	SEGMENT_ID	PACKET_TYPE	CONTROL_PACKET_TYPE/APP_NAME	SOURCE_ID	DESTINATION_ID	TRANSMITTER_ID	RECEIVER_ID
129	2	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
153	3	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
165	4	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
185	5	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
196	6	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
204	7	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
220	8	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
239	9	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
247	10	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
263	11	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
276	12	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
284	13	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
296	14	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
304	15	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
323	16	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
338	17	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
346	18	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
354	19	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
362	20	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7
373	21	0	Sensing	App1_SENSOR_APP	SENSOR-8	NODE-3	SENSOR-8	SENSOR-7