DIO Suppression Attack in RPL

Software: NetSim Standard v13.1 (64-bit), Visual Studio 2017/2019

Project Download Link:

https://github.com/NetSim-TETCOS/DIO_Suppression_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

In DIO Suppression Attack, a malicious node broadcast DIO message to legitimate nodes. If malicious node transmits repeatedly a DIO message that is considered consistent by the receiving nodes. If the nodes receive enough consistent DIOs, they will suppress their own DIO transmission. Since DIO messages are exploited to discover neighbours and the network topology, their continuous suppression can cause some nodes to remain hidden and some routes to remain undiscovered. DIO Suppression 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 transmits DIO message repeatedly to legitimate nodes.
- The legitimate nodes on listening to the malicious node DIO message they will suppress their own DIO transmission.
- The continuous suppression can cause some nodes to remain hidden and some routes to remain undiscovered.

The DIO.c 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_MaliciousNodeReplay(); //This function is used by the malicious node to transmit DIO message repeatedly to legitimate nodes.

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.

Settings that were done to create the network scenario for DIO Suppression Attack

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

2. Go to Your Work option in NetSim Home Screen and open the saved example, WITH_DIO_Suppression_Attack. The network scenario and the settings done is explained below:

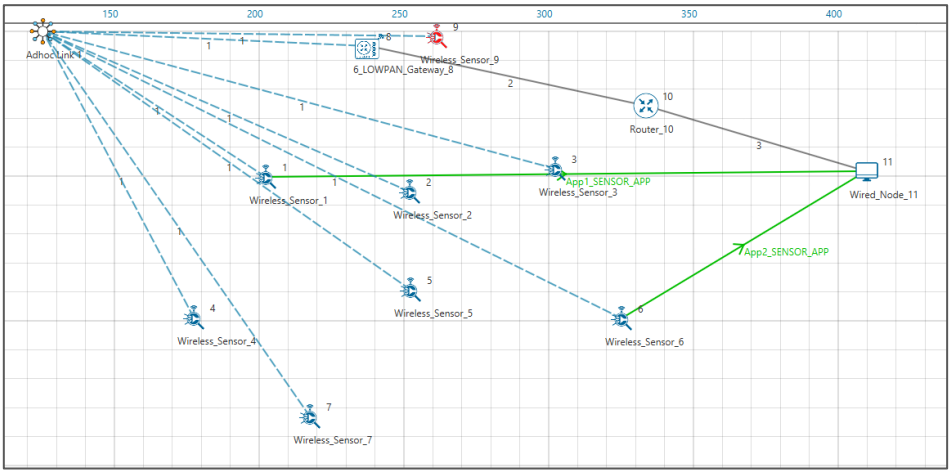

Figure 1: Network topology in this Project

Note: In above screenshot Red color Wireless_Sensor_Node_9 is a malicious node.

For Application 1:

- Source – Device id 1
- Destination – Device id 11
- Packet Size – 50Bytes
- Inter Arrival Time – 1000000microsec

For Application 2:

- Source – Device id 6
- Destination – Device id 11
- Packet Size – 50Bytes
- Inter Arrival Time – 1000000microsec

Link Properties (Adhoc Link 1)

- Channel Characteristics – Path Loss only
- Path Loss model – LOG DISTANCE
- Path Loss Exponent- 3

3. Go to 6LoWPAN Gateway Properties->Network_Layer-> DIORedundancyConstant-> 6.
   - The DIO suppression attack requires the adversary to transmit only k (DIO Redundancy Constant) DIO messages at each Trickle period.
   - DIO Redundancy Constant(k) acts as suppression threshold, as we set 6, the malicious node will replay the DIO message 6 times to the neighbouring nodes. After replaying the DIO message, the neighbouring nodes will suppress their own DIO transmission.
4. Run simulation and press any key to continue. NetSim simulation console will show the following message in the console “Waiting for NetSim MATLAB Interface to connect...”

5. Click on Options from NetSim design window and click on Open MATLAB Interface as shown in below given screenshot.

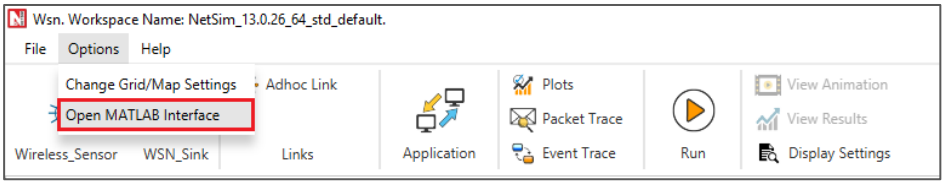

Figure 2: Open MATLAB Interface

6. It will open the popup related MATLAB Interfacing, Click on OK. As shown in below given screenshot

7. It will open MatlabInterface.exe console window. You will observe that as the simulation starts in NetSim, MATLAB gets initialized and the DODAG plot associated with the IoT network is plotted during runtime.

8. View the packet animation. You will find that malicious node (Device id 9) even after receiving DIO message from neighbour nodes it will start transmitting repeatedly DIO message to neighbour nodes.

9. This will cause some nodes to remain hidden and some route to remain undiscovered or in the worst case, a partition of the network.

Settings that were done to create the network scenario for WITHOUT_DIO Suppression Attack

To run simulations without DIO Suppression attack set the value of the variable DIO_ATTACK_ENABLE to 0 instead of 1. Rebuild the RPL source codes and run Simulation.

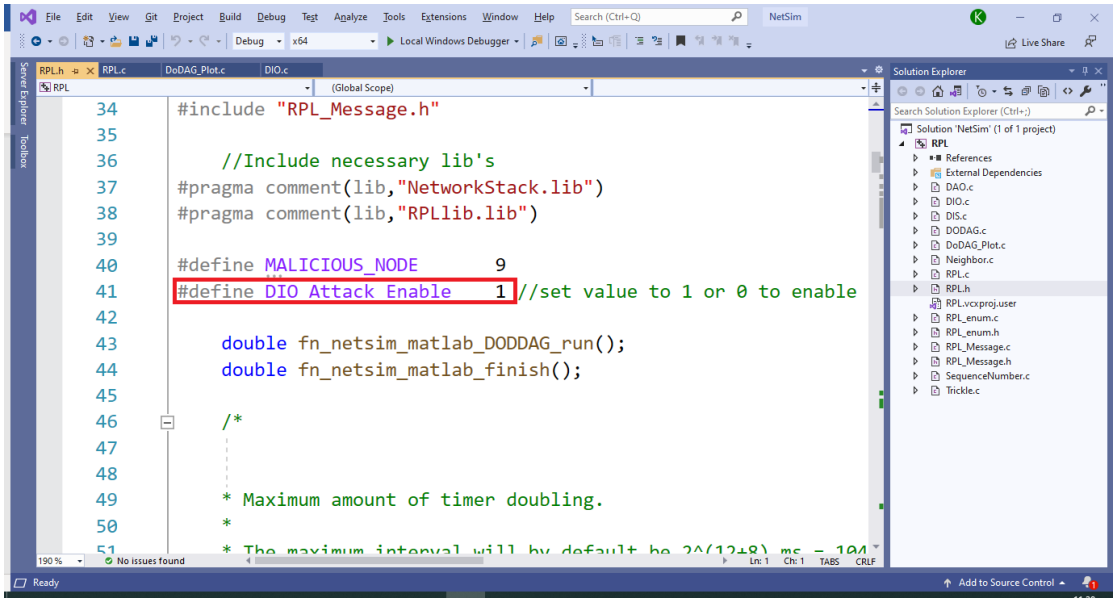

Figure 3: DIO_ATTACK_ENABLE to set 0 instead of 1

Results and discussion

Case 1: With DIO Suppression Attack

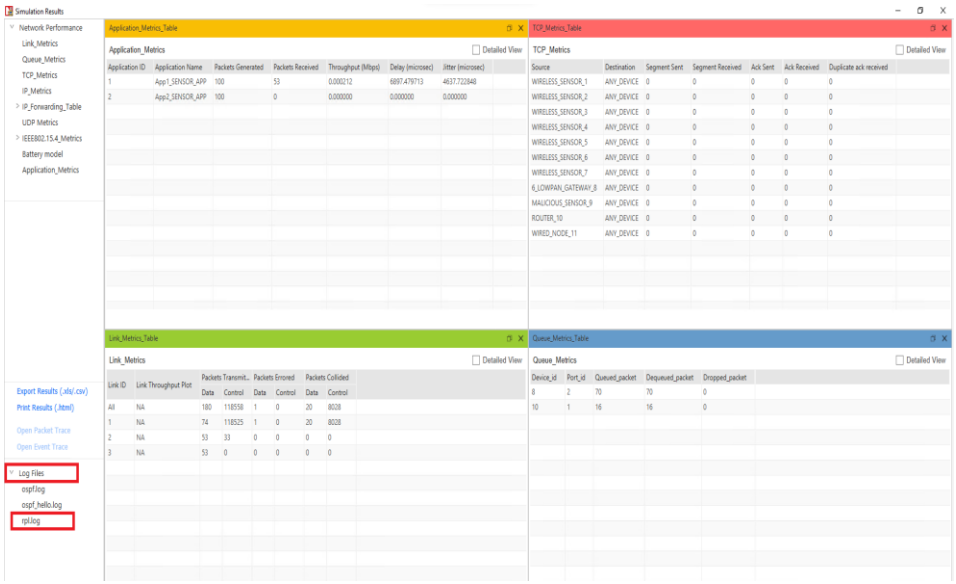

Figure 4: NetSim results dashboard with throughput highlighted

DODAG Formation Graph:

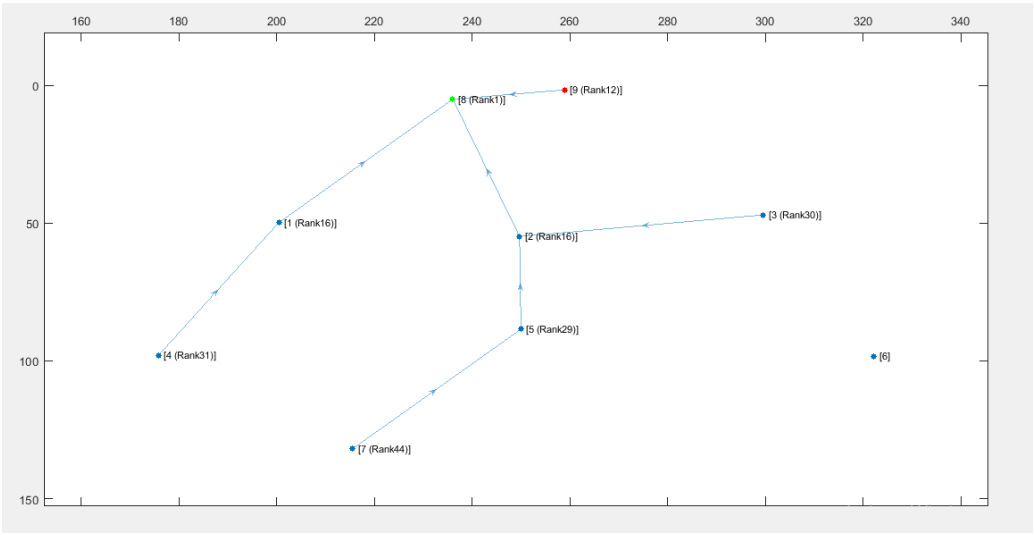

Figure 5: DODAG Formation Graph from MATLAB

When root node (Wireless_Sensor_Node_8) broadcast the DIO message all nodes that are present in the communication range will also broadcast their own DIO messages but when malicious node broadcasts the DIO message, it will repeatedly transmit the DIO message to the neighbour nodes such that it prevents the DIO messages from other neighbour nodes reaching them.

So, it degrades the routing information, and some nodes remain hidden in the network.
We can observe from the above graph that Wireless_Sensor_Node_6 is not part of DODAG formation as it is not discovered and remain hidden in the network.

Case 2: Without DIO Suppression Attack

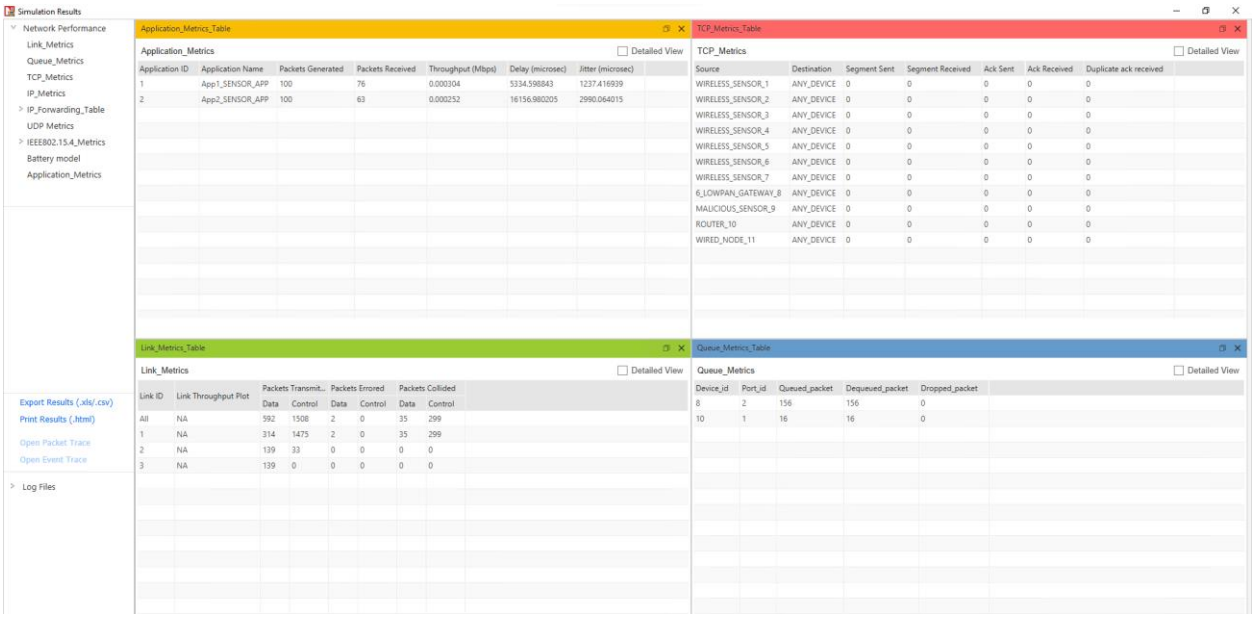

Figure 6: NetSim results dashboard with throughput highlighted

DODAG Formation Graph:

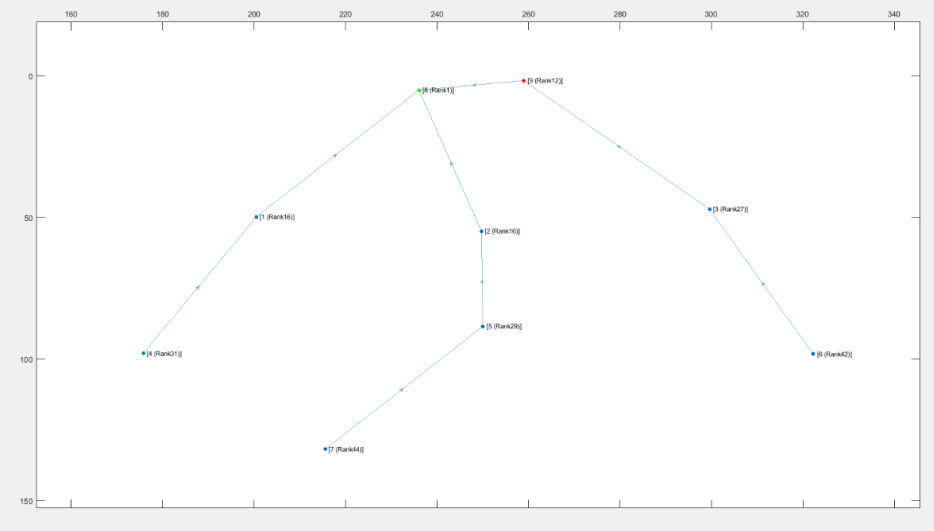

Figure 7: DODAG Formation Graph from MATLAB

Case 3: With DIO Suppression Attack

The DIORedundancyConstant is set to 7 in the following case.

DODAG Formation Graph:

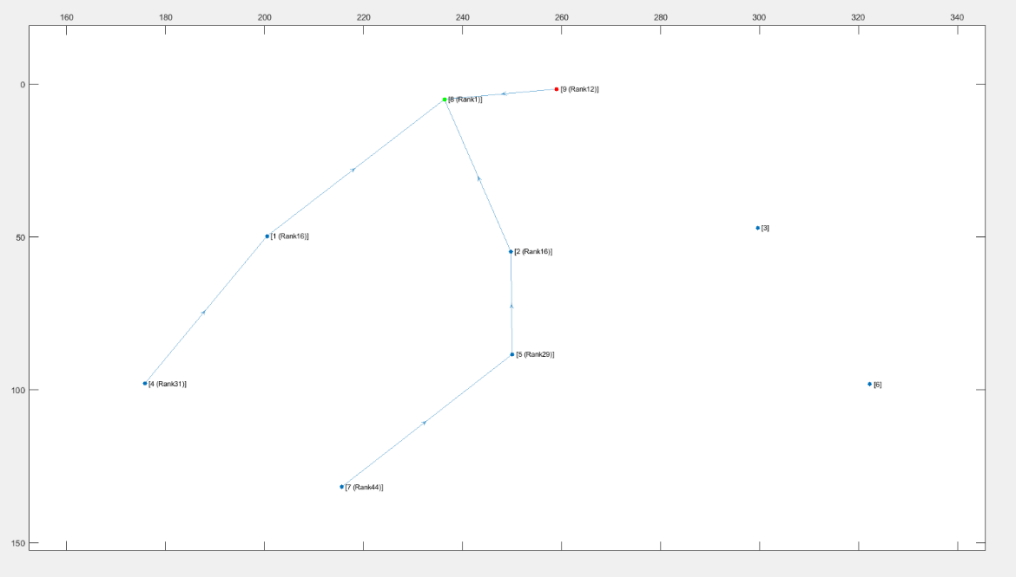

Figure 8: DODAG Formation Graph from MATLAB With DIORedundancyConstant is set to 7

We can observe from the graph that Wireless_Sensor_Node_3 and Wireless_Sensor_Node_6 is not part of DODAG formation as it is not discovered and remain hidden in the network.

We can observe from the Simulation result dashboard that when we enable DIO Suppression attack in that situation some nodes are hidden due to which our throughput is getting decreased.

DIO Suppression severely degrade the performance of Low Power and Lossy Network (LLNs) because of the repeatedly transmitting the DIO message by the malicious node to neighbouring nodes.

The DIO suppression attack, an attack that induces victim nodes to suppress the transmission of DIO messages. This causes a general degradation of the routes quality that can lead, eventually, to network partitions.

Appendix: NetSim source code modifications and steps.

1. Add the following MATLAB install directory path in the Environment PATH variable <MATLAB_INSTALL_DIRECTORY>\bin\win64

   For eg: C:\Program Files\MATLAB\R2020a\bin\win64

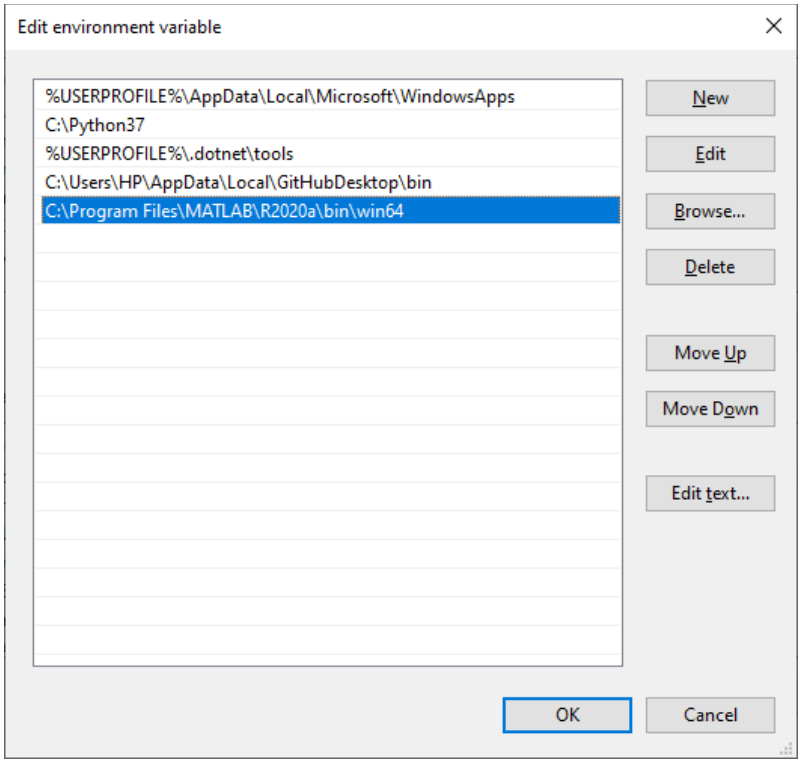

Figure 9: Set environment variable path

Note: If the machine has more than one MATLAB installed, the directory for the target platform must be ahead of any other MATLAB directory (for instance, when compiling a 64-bit application, the directory in the MATLAB 64-bit installation must be the first one on the PATH).

2. Open Command prompt as admin and execute the command “matlab -regserver”. This will register MATLAB as a COM automation server and is required for NetSim to start MATLAB automation server during runtime.
3. Go to home page, Click on Your work>Source Code and click on the Open code button.
4. Set malicious node id in RPL.h file.
   #define MALICIOUS NODE 9

The section of code that is highlighted in red color is added to the RPL_Message.c file under rpl_process_ctrl_msg() function.

```
Switch(pstruEventDetails->pPacket->nControlDataType % 100)
{
Case DODAG_Information_Object:
#if DIO_Attack_Enable
```

```
if(fn_NetSim_RPL_MaliciousNode(pstruEventDetails))
{
rpl_process_dio_msg();
Fn_NetSim_RPL_MaliciousNodeReplay(pstruEventDetails);
}
else
{
rpl_process_dio_msg();
}
#else
                rpl_process_dio_msg();
#elseif
break:
```

5. Now right click on Solution explorer and select Rebuild.
   a. Upon rebuilding, libRPL.dll will automatically get replaced in the respective bin folders of the current workspace.