Sink Hole Attack using RPL in IOT

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

Project Download Link:
https://github.com/NetSim-TETCOS/SinkHole_attack_in_IoT_RPL_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 sinkhole Attack, a compromised node or malicious node advertises fake rank information to form the fake routes. After receiving the message packet, it drops the packet information. Sinkhole attacks affect the performance of IoT networks protocols such as RPLprotocol.

Implementation in RPL (for 1 sink)

- In RPL the transmitter broadcasts the DIO during DODAGformation.
- 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 fakerank.
- The other node on listening to the malicious node DIO message the update their rank according to the fakerank.
- 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 zerothroughput.

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

- fn_NetSim_RPL_MaliciousNode(); //This function is used to identify whether a current device is malicious or not in-order to establish malicious behavior.
- fn_NetSim_RPL_MaliciousRank(); //This function is used to give a fake rank to the malicious node.
- 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 rankfn_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 to simualte

1. Open the Source codes in Visual Studio by going to Your work-> Source code and Clicking on Open code button in NetSim Home Screen window.
2. Now right click on Solution explorer and select Rebuild.

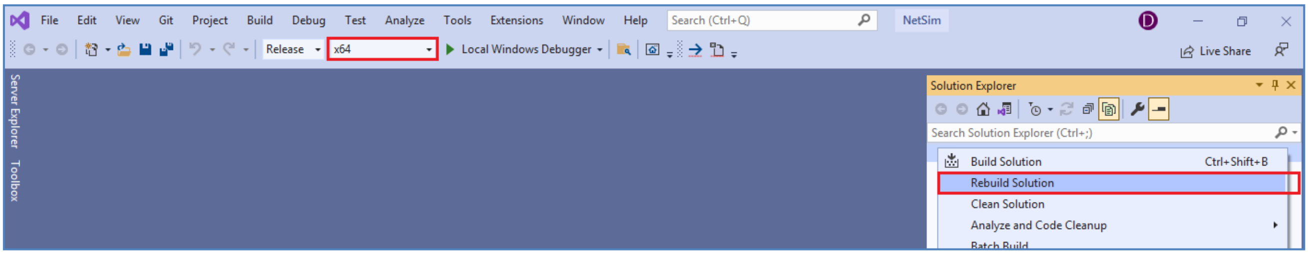

Fig 1: Screenshot of NetSim project source code in Visual Studio

3. Upon rebuilding, modified binarieswill automatically get updated in the respective bin folders of the currentworkspace.

Example

1. The **WorkSpace_SinkHole_Attack_RPL** 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 **SinkHole_Attack_in_RPL_Example** from the list of experiments.
2. The saved network scenario consists of
    a. 5 Wireless Sensor
    b. 1 6_LOWPAN Gateway
    c. 1 Router
    d. 1 Wired Node

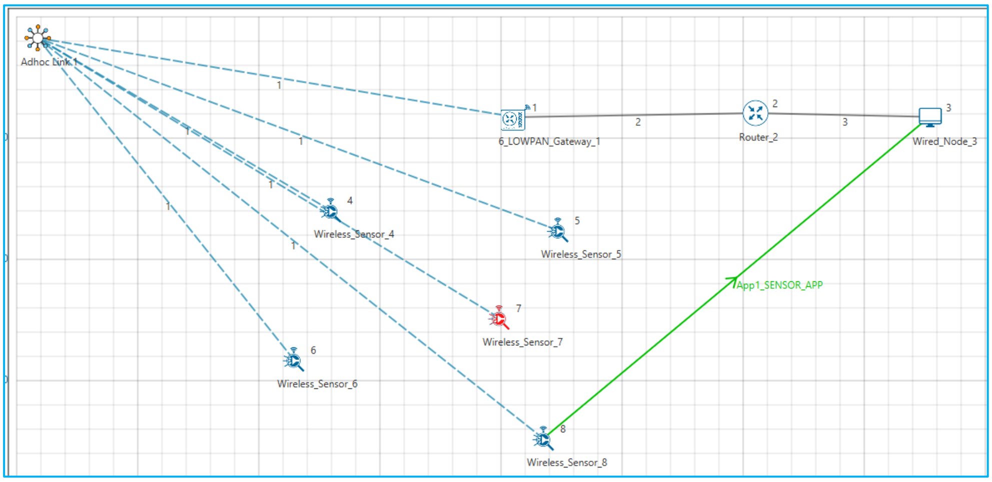

Fig 2: IoT Network Topology

- Channel Characteristics: Path Loss Only, Path Loss Model: Log Distance, Path Loss Exponent: 2
- Run Simulation for 100 Seconds.

Results and discussion

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)

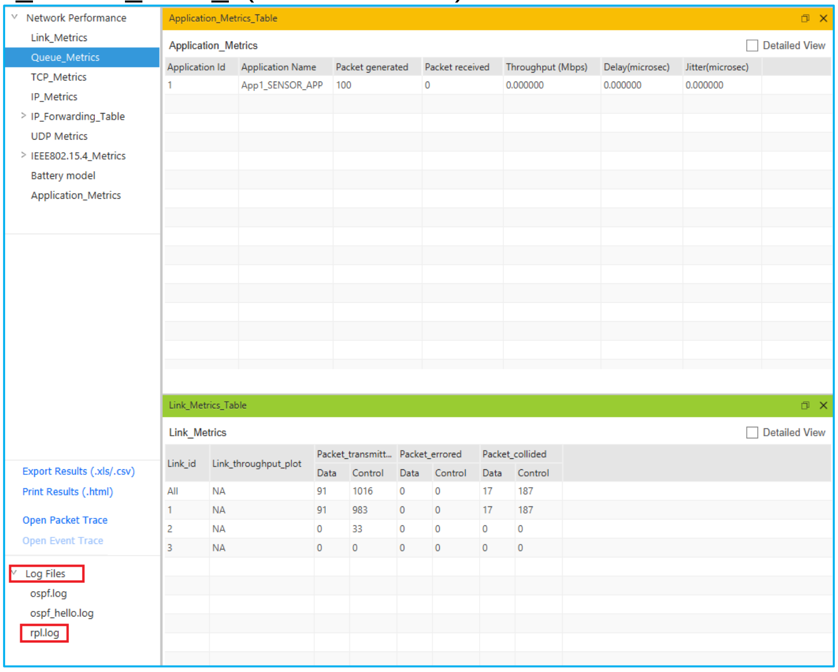

Fig 3**:** From the Result Dashboard window

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

Fig 4**:**NetSim Packet Trace

Introducing multiple malicious nodes:

To introduce the multiple malicious nodes in the network, consider a larger network consisting of more of sensors and with multiple sensor devices generating traffic. Malicious nodes can be distributed in different locations of the network and their impact on the network can be analyzed.

1. Add one more sensor i.e., Sensor_9 for the similar scenario and create traffic as shown below.

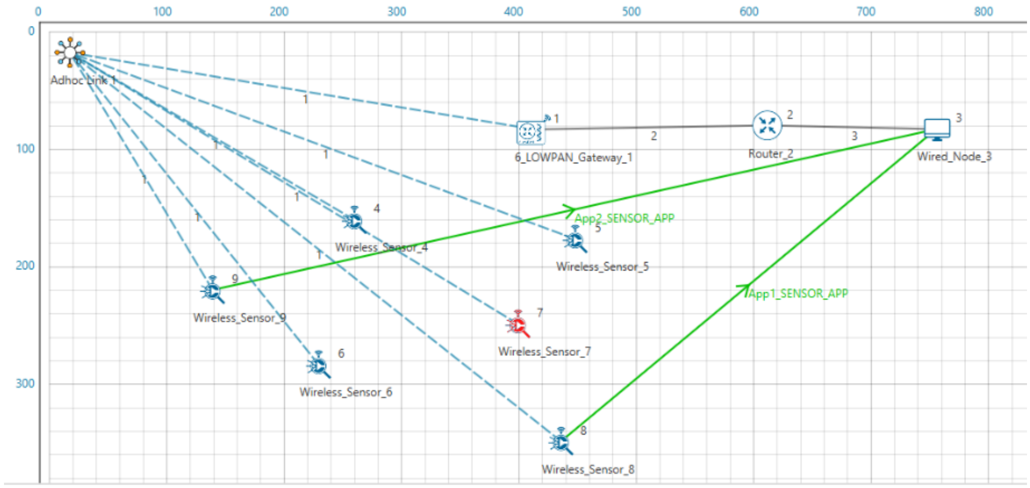

Fig 5:IoT Network Topology for multiple malicious nodes

2. Consider sensor 6 and 7 as malicious nodes with fake rank by defining it in the Malicious.c file as shown below.

Malicious.c*

RPL (Global Scope)

```
#include "main.h"
#include "RPL.h"
#include "RPL_enum.h"
#define MALICIOUS_NODE1 7
#define MALICIOUS_RANK1 3

#define MALICIOUS_NODE2 6
#define MALICIOUS_RANK2 4

/**
Function prototypes
*/
int fn_NetSim_RPL_MaliciousNode(NetSim_EVENTDETAILS*);
void fn_NetSim_RPL_MaliciousRank(NetSim_EVENTDETAILS*);
void rpl_drop_msg();
int fn_NetSim_RPL_FreePacket(NetSim_PACKET*);
```

Fig 6: Defining the malicious nodes in Malicious.c file

3. In fn_NetSim_RPL_MaliciousNode() function, the if condition for checking malicious nodes needs to be updated.

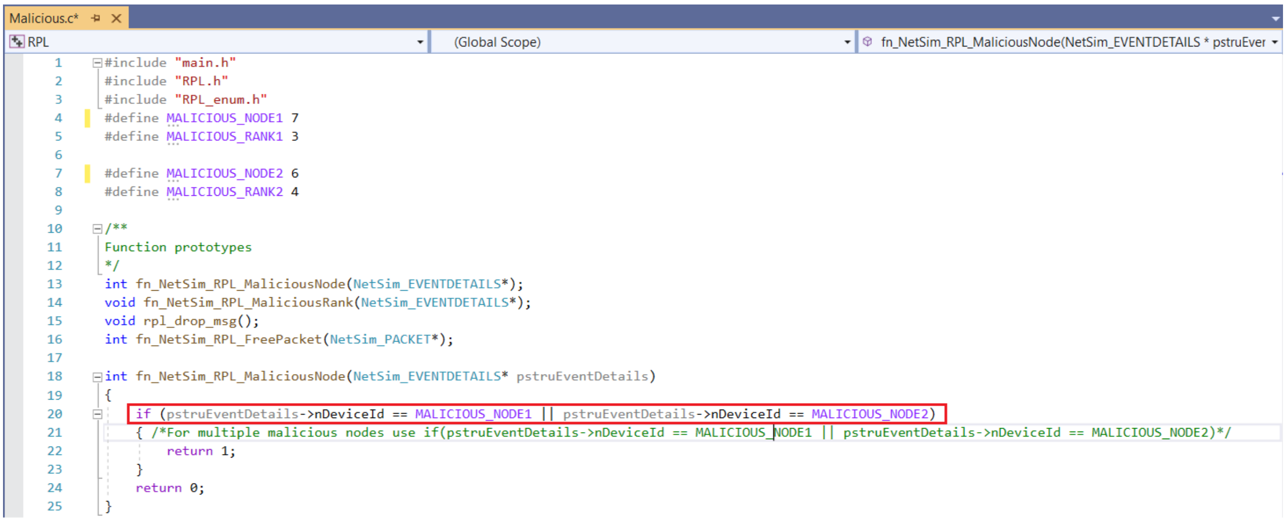

Fig 7: If condition for checking multiple malicious node

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

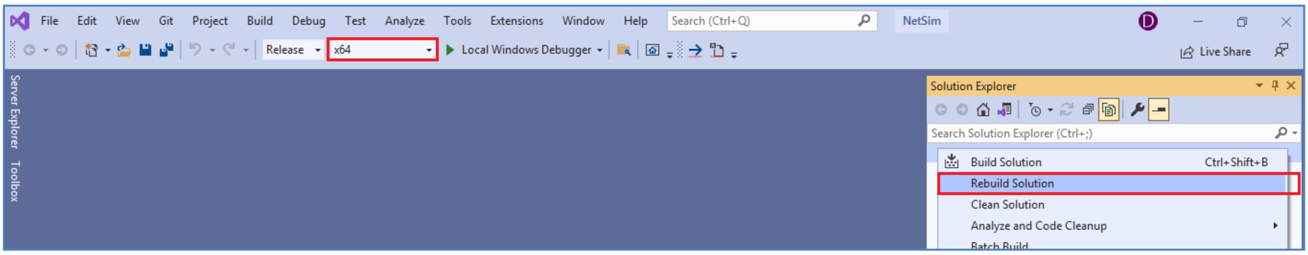

Fig 8: Rebuild solution explorer

Results and discussion

Sensor 8 will consider sensor 7 as a parent and sensor 9 will consider sensor 6 as parent instead of sensor 4 since sensor 6 advertises lower rank compared to sensor 4. Packets reach sensors 7 and 6 get dropped. Results can be visualized in the rpllog.txt and packet trace.

You can also check the distribution of ranks with the help of DODAG visualizer- https://support.tetcos.com/en/support/solutions/articles/14000134056-how-to-visualize-the-rpl-dodag-in-netsim-iot-simulations-
The DoDAG plots appear vertically flipped when compared to the network topology in NetSim since the origin (0,0) is at the top left in NetSim whereas it is in the bottom left in the plot window.

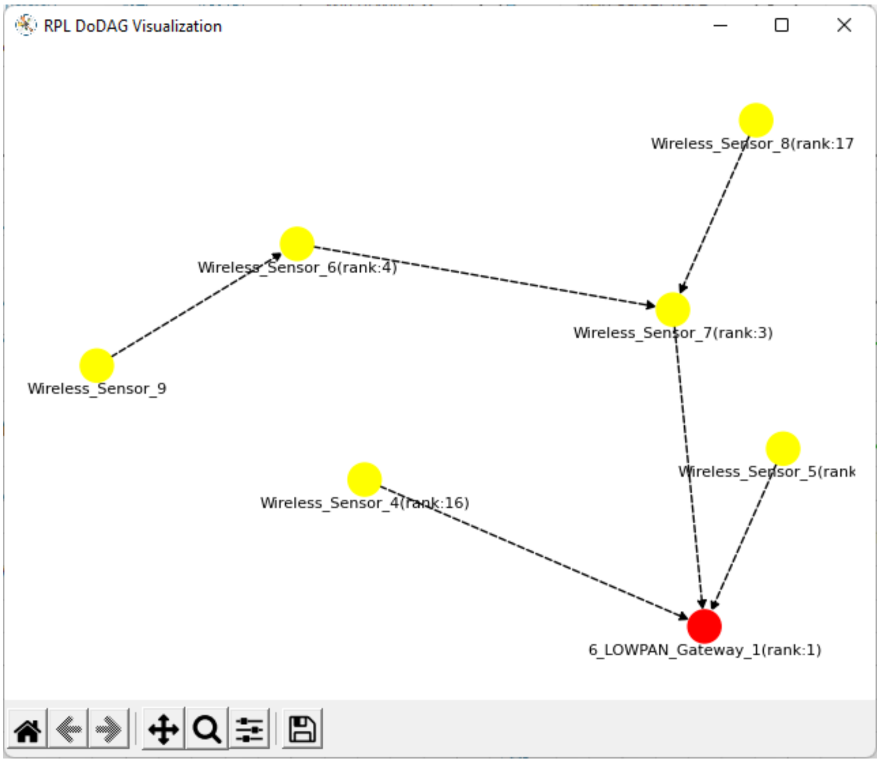

Fig 9: RPL DODAG Visualizer

Note: Conditions for malicious node to be able to attract other legitimate nodes:

- *The malicious node should be within the range of other nodes.*
- *The malicious nodes' DIO broadcast should be received by other nodes with a rank lower than the other DIO messages received.*

Appendix: NetSim source code modifications

Set malicious node id and the fakeRank in malicious.c file which is present under RLP Project.

```
#include "main.h"
#include "RPL.h"
#include "RPL_enum.h"
#define MALICIOUS_NODE1 7
#define MALICIOUS_RANK1 3

#define MALICIOUS_NODE2 4
#define MALICIOUS_RANK2 4
```

Changes code to fn_NetSim_RPL_Run(), in RPL.c file, **within RPL project**

```
_declspec (dllexport) int fn_NetSim_RPL_Run()
{
switch (pstruEventDetails->nEventType)
{
case NETWORK_OUT_EVENT:
{
}
break;
case NETWORK_IN_EVENT:
{
rpl_add_to_neighbor_list();
if (is_rpl_control_packet(pstruEventDetails->pPacket))
{
if (fn_NetSim_RPL_MaliciousNode(pstruEventDetails))
fn_NetSim_RPL_MaliciousRank(pstruEventDetails);
else
rpl_process_ctrl_msg();
fn_NetSim_Packet_FreePacket(pstruEventDetails->pPacket);
pstruEventDetails->pPacket = NULL;
}
else if (pstruEventDetails->nPacketId&&fn_NetSim_RPL_MaliciousNode(pstruEventDetails))
{
rpl_drop_msg();
}
}
break;
```