ARP Spoofing Implementation in NetSim

Software: NetSim Standard v14.3, Visual Studio 2022

Project Download Link:

https://github.com/NetSim-TETCOS/ARP-Spoofing-v14.3/archive/refs/heads/main.zip

Follow the instructions specified in the following link to download and set up the Project in NetSim:

https://support.tetcos.com/en/support/solutions/articles/14000128666-downloading-and-setting-up-netsim-fileexchange-projects

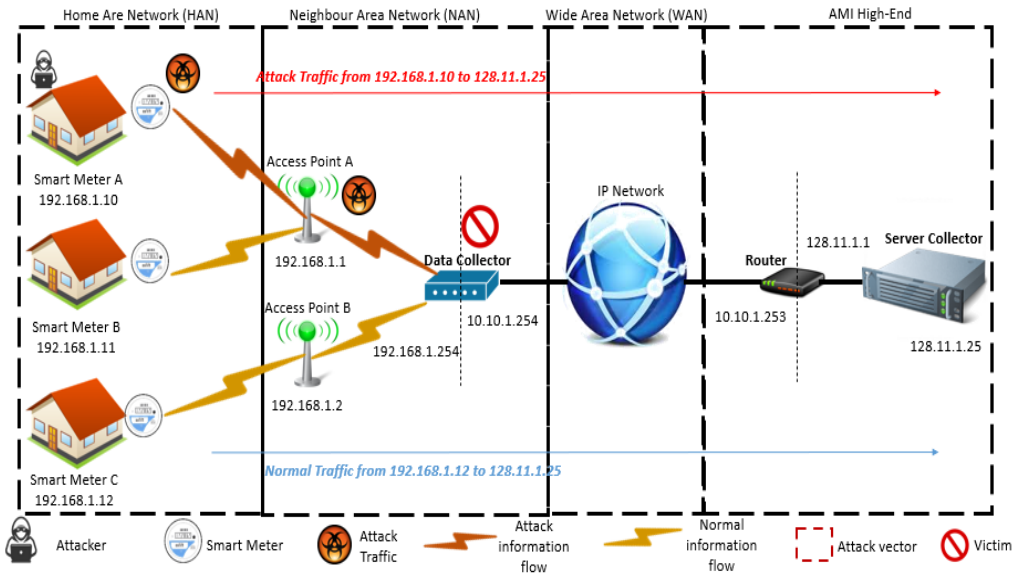
Introduction:

ARP spoofing (also known as ARP poisoning) is a type of cyber-attack where an attacker sends fake ARP messages onto a local network. The goal is to associate the attacker's MAC address with the IP address of another device, effectively tricking other devices into sending traffic to the attacker instead of the intended recipient.

How ARP Spoofing Works:

1. Normal ARP Behaviour: Devices on a LAN use ARP to map IP addresses to MAC addresses.
2. Spoofing: The attacker sends forged ARP replies to the victim/gateway, associating the attacker's MAC with the IP of the gateway or target.

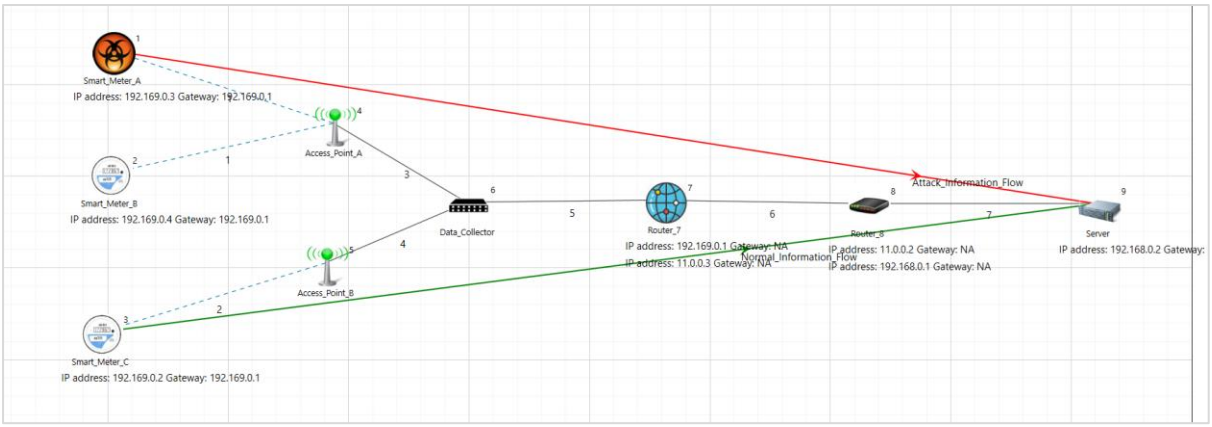
Example case:



Here, Smart Meter A attempting False Data Injection Attack to falsify data from Smart Meter C to Server Collector.

To initiate this attack, Smart Meter A will have to poison Data Collector to spoof the IP address of Smart Meter C and send falsified traffic to Server Collector as Smart Meter C.

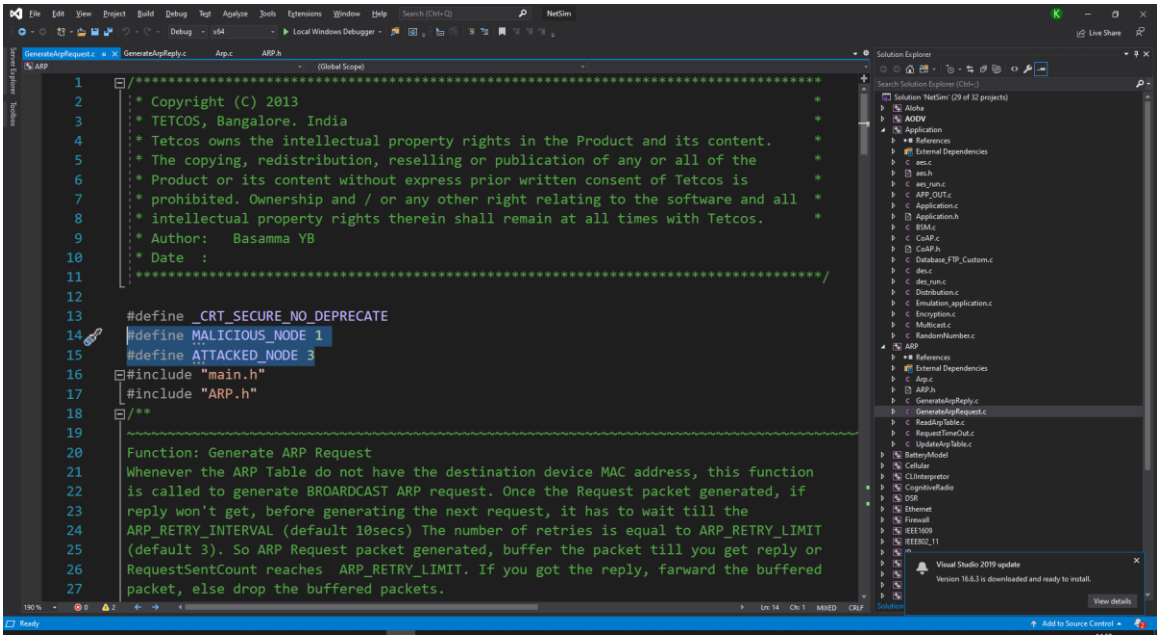
Network Scenario:



Source code changes:

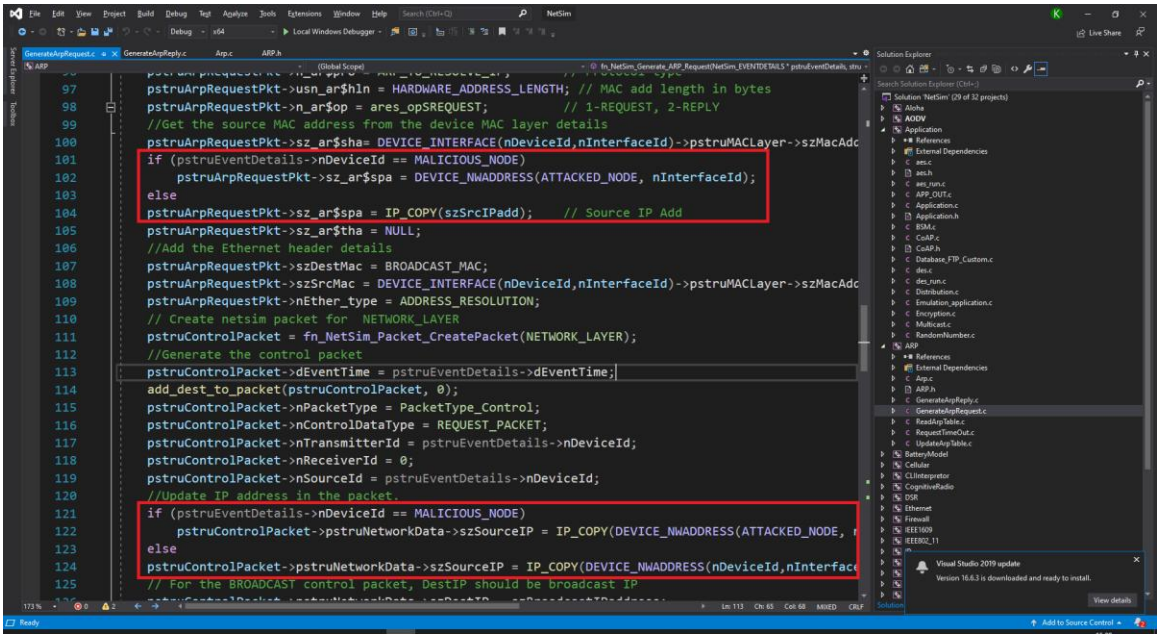
Source codes of ARP protocol is modified

- The attacker/malicious node (Smart Meter A) and the legitimate/attacked node (Smart Meter C) are defined in the GenerateArpRequest.c file of the ARP project.
- The attacker node sends falsified ARP packets mapping the attackers MAC address with IP address of Smart Meter C.



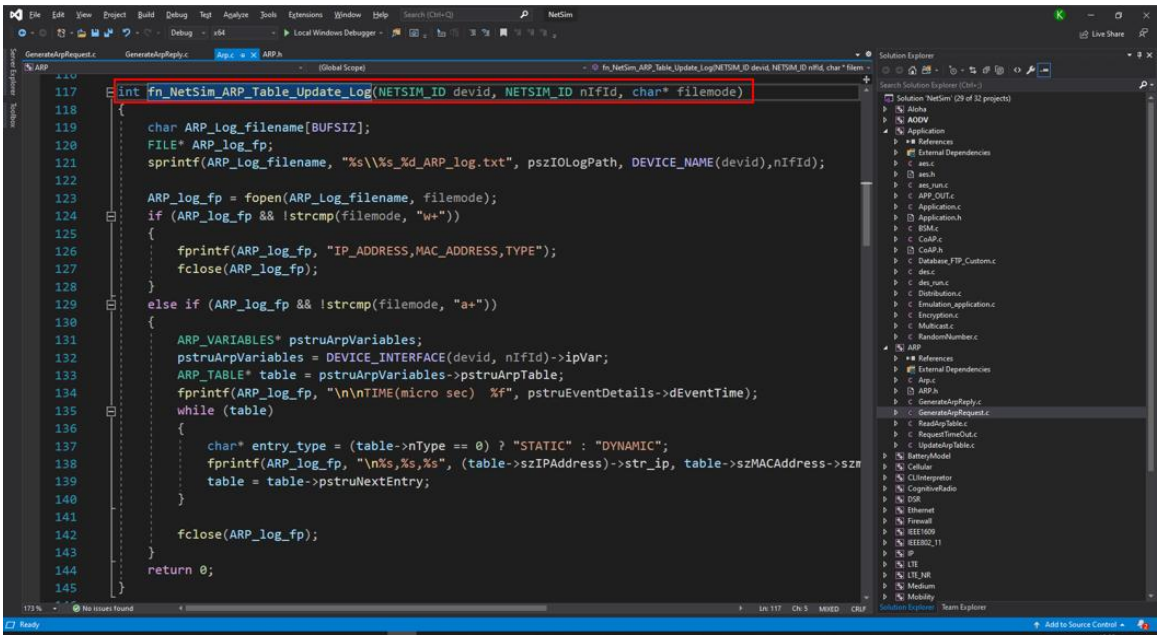
```
/*****************************************************************************
 * Copyright (C) 2013                                                        *
 * TETCOS, Bangalore. India                                                  *
 * Tetcos owns the intellectual property rights in the Product 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:    Basamma YB                                                     *
 * Date   :                                                                  *
 *****************************************************************************/

#define _CRT_SECURE_NO_DEPRECATE
#define MALICIOUS_NODE 1
#define ATTACKED_NODE 3
#include "main.h"
#include "ARP.h"
/**
Function: Generate ARP Request
Whenever the ARP Table do not have the destination device MAC address, this function
is called to generate BROARDCAST ARP request. Once the Request packet generated, if
reply won't get, before generating the next request, it has to wait till the
ARP_RETRY_INTERVAL (default 10secs) The number of retries is equal to ARP_RETRY_LIMIT
(default 3). So ARP Request packet generated, buffer the packet till you get reply or
RequestSentCount reaches  ARP_RETRY_LIMIT. If you got the reply, farward the buffered
packet, else drop the buffered packets.
```

- Further, in the function fn_NetSim_Generate_ARP_Request() which is part of the GenerateArpRequest.c file, code has been added to include a false IP address in the ARP packet generated from the attacker/malicious node.



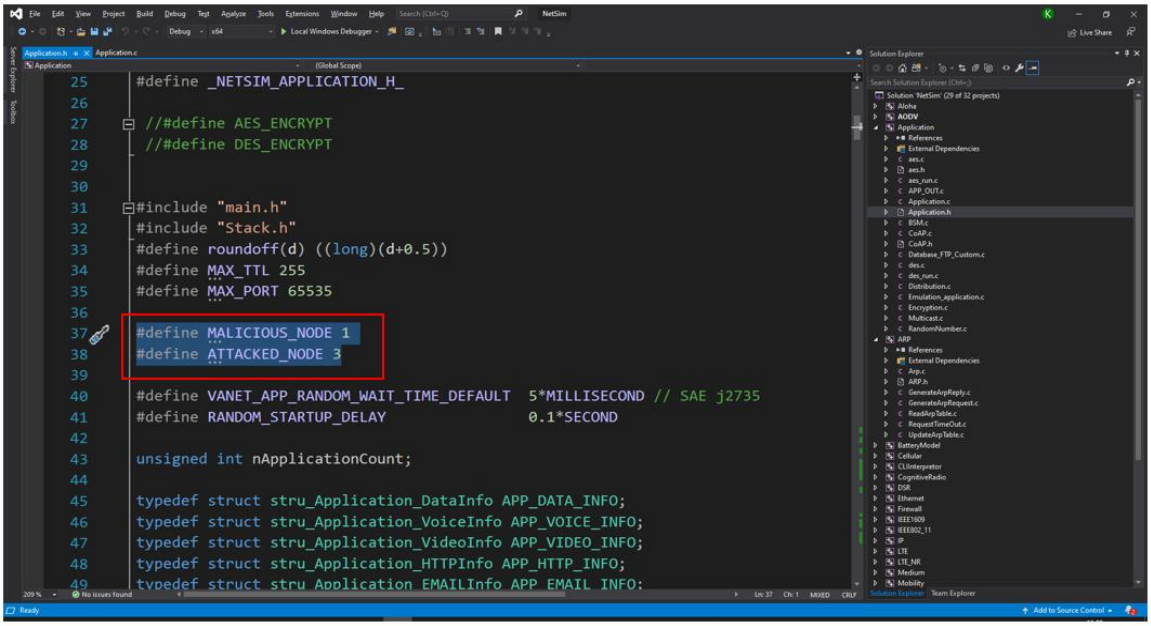
```
97  pstruArpRequestPkt->usn_ar$hln = HARDWARE_ADDRESS_LENGTH; // MAC add length in bytes
98  pstruArpRequestPkt->n_ar$op = ares_opSREQUEST;          // 1-REQUEST, 2-REPLY
99  //Get the source MAC address from the device MAC layer details
100 pstruArpRequestPkt->sz_ar$sha= DEVICE_INTERFACE(nDeviceId,nInterfaceId)->pstruMACLayer->szMacAdc
101 if (pstruEventDetails->nDeviceId == MALICIOUS_NODE)
102     pstruArpRequestPkt->sz_ar$spa = DEVICE_NWADDRESS(ATTACKED_NODE, nInterfaceId);
103 else
104 pstruArpRequestPkt->sz_ar$spa = IP_COPY(szSrcIPadd);    // Source IP Add
105 pstruArpRequestPkt->sz_ar$tha = NULL;
106 //Add the Ethernet header details
107 pstruArpRequestPkt->szDestMac = BROADCAST_MAC;
108 pstruArpRequestPkt->szSrcMac = DEVICE_INTERFACE(nDeviceId,nInterfaceId)->pstruMACLayer->szMacAdc
109 pstruArpRequestPkt->nEther_type = ADDRESS_RESOLUTION;
110 // Create netsim packet for  NETWORK_LAYER
111 pstruControlPacket = fn_NetSim_Packet_CreatePacket(NETWORK_LAYER);
112 //Generate the control packet
113 pstruControlPacket->dEventTime = pstruEventDetails->dEventTime;
114 add_dest_to_packet(pstruControlPacket, 0);
115 pstruControlPacket->nPacketType = PacketType_Control;
116 pstruControlPacket->nControlDataType = REQUEST_PACKET;
117 pstruControlPacket->nTransmitterId = pstruEventDetails->nDeviceId;
118 pstruControlPacket->nReceiverId = 0;
119 pstruControlPacket->nSourceId = pstruEventDetails->nDeviceId;
120 //Update IP address in the packet.
121 if (pstruEventDetails->nDeviceId == MALICIOUS_NODE)
122     pstruControlPacket->pstruNetworkData->szSourceIP = IP_COPY(DEVICE_NWADDRESS(ATTACKED_NODE,
123 else
124 pstruControlPacket->pstruNetworkData->szSourceIP = IP_COPY(DEVICE_NWADDRESS(nDeviceId,nInterfac
125 // For the BROADCAST control packet, DestIP should be broadcast IP
```

- In the ARP.c file, an additional function fn_NetSim_ARP_Table_Update_Log() has been added to generate ARP table logs for each node in the network that runs the ARP protocol. This function is called from various parts of the ARP protocol source code where ARP table updates occur.



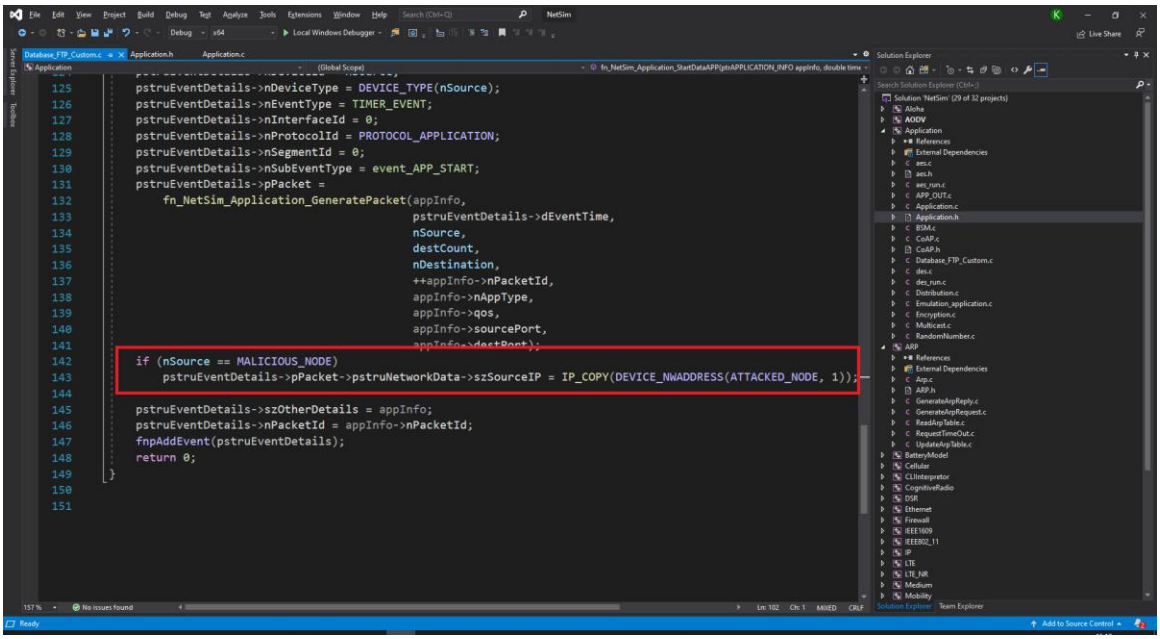
```
117 int fn_NetSim_ARP_Table_Update_Log(NETSIM_ID devid, NETSIM_ID nIfId, char* filemode)
118 {
119     char ARP_Log_filename[BUFSIZ];
120     FILE* ARP_log_fp;
121     sprintf(ARP_Log_filename, "%s\\%s_%d_ARP_log.txt", pszIOLogPath, DEVICE_NAME(devid),nIfId);
122
123     ARP_log_fp = fopen(ARP_Log_filename, filemode);
124     if (ARP_log_fp && !strcmp(filemode, "w+"))
125     {
126         fprintf(ARP_log_fp, "IP_ADDRESS,MAC_ADDRESS,TYPE");
127         fclose(ARP_log_fp);
128     }
129     else if (ARP_log_fp && !strcmp(filemode, "a+"))
130     {
131         ARP_VARIABLES* pstruArpVariables;
132         pstruArpVariables = DEVICE_INTERFACE(devid, nIfId)->ipVar;
133         ARP_TABLE* table = pstruArpVariables->pstruArpTable;
134         fprintf(ARP_log_fp, "\n\nTIME(micro sec)  %f", pstruEventDetails->dEventTime);
135         while (table)
136         {
137             char* entry_type = (table->nType == 0) ? "STATIC" : "DYNAMIC";
138             fprintf(ARP_log_fp, "\n%s,%s,%s", (table->szIPAddress)->str_ip, table->szMACAddress->szm
139             table = table->pstruNextEntry;
140         }
141
142         fclose(ARP_log_fp);
143     }
144     return 0;
145 }
```

- The source code of the Application project was modified to enable the attacker/malicious node to send spoofed data packets using the IP address of the legitimate/attacked node as the source IP address. The attacker/malicious node (Smart Meter A) and the legitimate/attacked node (Smart Meter C) are defined in the Application.h file of the Application project



```
#define _NETSIM_APPLICATION_H_

//#define AES_ENCRYPT
//#define DES_ENCRYPT


#include "main.h"
#include "Stack.h"
#define roundoff(d) ((long)(d+0.5))
#define MAX_TTL 255
#define MAX_PORT 65535

#define MALICIOUS_NODE 1
#define ATTACKED_NODE 3

#define VANET_APP_RANDOM_WAIT_TIME_DEFAULT  5*MILLISECOND // SAE j2735
#define RANDOM_STARTUP_DELAY                0.1*SECOND

unsigned int nApplicationCount;

typedef struct stru_Application_DataInfo APP_DATA_INFO;
typedef struct stru_Application_VoiceInfo APP_VOICE_INFO;
typedef struct stru_Application_VideoInfo APP_VIDEO_INFO;
typedef struct stru_Application_HTTPInfo APP_HTTP_INFO;
typedef struct stru_Application_EMAILInfo APP_EMAIL_INFO;
```

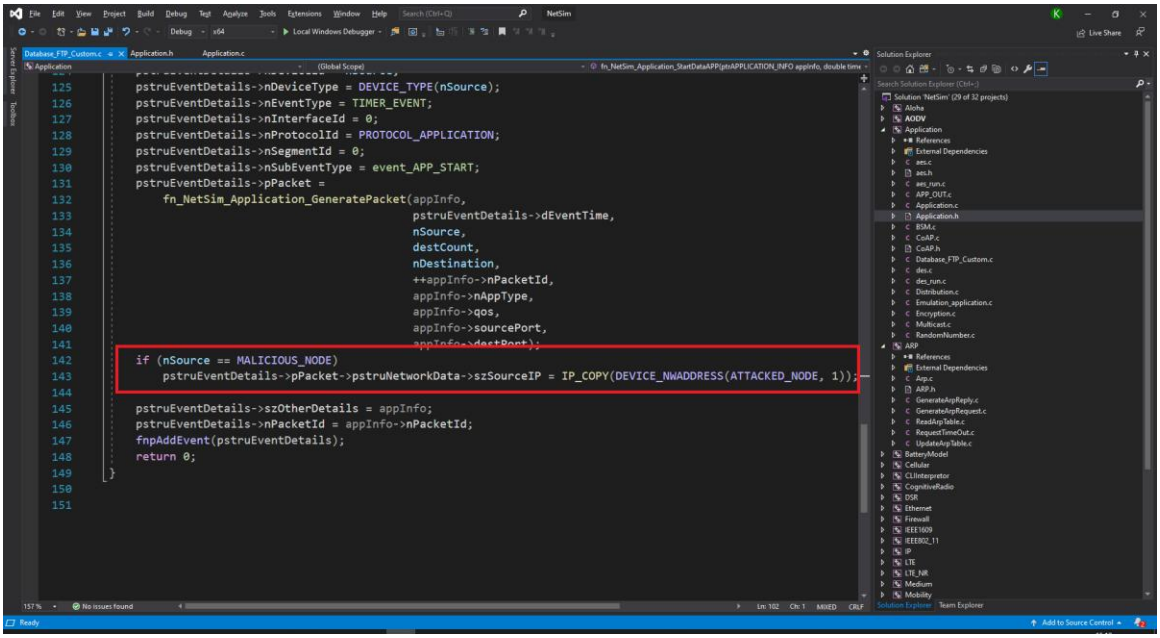
- In the function fn_NetSim_Application_StartDataAPP(), which is part of the Database_FTP_Custom.c file, code was added to include a false IP address in the data packet generated from the attacker/malicious node.



```
    pstruEventDetails->nDeviceType = DEVICE_TYPE(nSource);
    pstruEventDetails->nEventType = TIMER_EVENT;
    pstruEventDetails->nInterfaceId = 0;
    pstruEventDetails->nProtocolId = PROTOCOL_APPLICATION;
    pstruEventDetails->nSegmentId = 0;
    pstruEventDetails->nSubEventType = event_APP_START;
    pstruEventDetails->pPacket =
        fn_NetSim_Application_GeneratePacket(appInfo,
                                             pstruEventDetails->dEventTime,
                                             nSource,
                                             destCount,
                                             nDestination,
                                             ++appInfo->nPacketId,
                                             appInfo->nAppType,
                                             appInfo->qos,
                                             appInfo->sourcePort,
                                             appInfo->destPort);

    if (nSource == MALICIOUS_NODE)
        pstruEventDetails->pPacket->pstruNetworkData->szSourceIP = IP_COPY(DEVICE_NWADDRESS(ATTACKED_NODE, 1));

    pstruEventDetails->szOtherDetails = appInfo;
    pstruEventDetails->nPacketId = appInfo->nPacketId;
    fnpAddEvent(pstruEventDetails);
    return 0;
}
```

- In the function fn_NetSim_Application_StartDataAPP(), which generates the first packet of an application and is part of the Database_FTP_Custom.c file, code was added to include a false IP address in the data packet generated from the attacker/malicious node.

- Further, in the function fn_NetSim_Application_GenerateNextPacket(), which generates subsequent packets of an application and is part of the Application.c file, similar code was added to include a false IP address in the data packets generated from the attacker/malicious node.



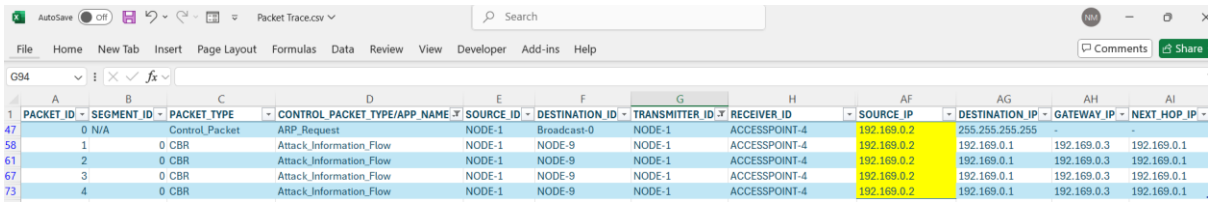
```
125     pstruEventDetails->nDeviceType = DEVICE_TYPE(nSource);
126     pstruEventDetails->nEventType = TIMER_EVENT;
127     pstruEventDetails->nInterfaceId = 0;
128     pstruEventDetails->nProtocolId = PROTOCOL_APPLICATION;
129     pstruEventDetails->nSegmentId = 0;
130     pstruEventDetails->nSubEventType = event_APP_START;
131     pstruEventDetails->pPacket =
132         fn_NetSim_Application_GeneratePacket(appInfo,
133                                              pstruEventDetails->dEventTime,
134                                              nSource,
135                                              destCount,
136                                              nDestination,
137                                              ++appInfo->nPacketId,
138                                              appInfo->nAppType,
139                                              appInfo->qos,
140                                              appInfo->sourcePort,
141                                              appInfo->destPort);
142     if (nSource == MALICIOUS_NODE)
143         pstruEventDetails->pPacket->pstruNetworkData->szSourceIP = IP_COPY(DEVICE_NWADDRESS(ATTACKED_NODE, 1));
144
145     pstruEventDetails->szOtherDetails = appInfo;
146     pstruEventDetails->nPacketId = appInfo->nPacketId;
147     fnpAddEvent(pstruEventDetails);
148     return 0;
149 }
150
151
```

Network setting:

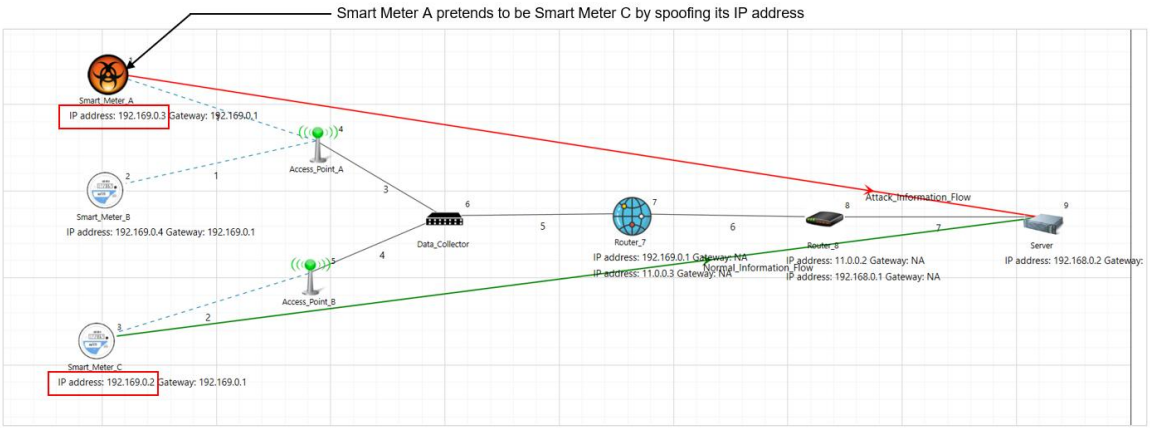
- Enable packet trace prior to the simulation, by clicking on configure reports on top ribbon and disable the static ARP table by clicking on Options > Static ARP > Disable.
- Run the simulation for 10 seconds.

Results:

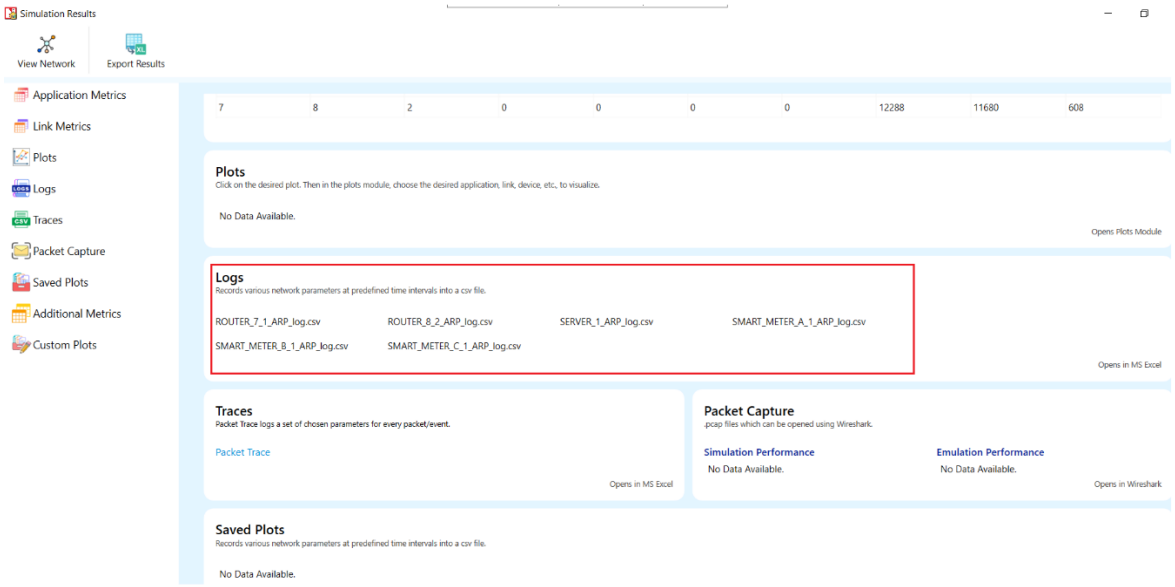
The screenshot below displays the NetSim packet trace with packets filtered to show Smart Meter A (the attacker node) as the transmitter. Both the ARP request and the data packets sent from the attacker node have a spoofed source IP address (192.169.0.2) instead of its actual IP address (192.169.0.3).

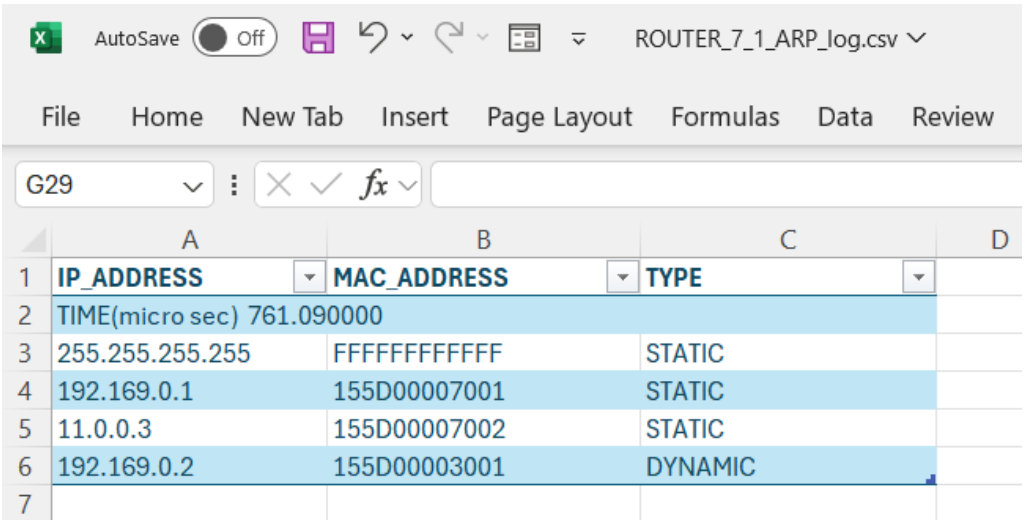


	A	B	C	D	E	F	G	H	AF	AG	AH	AI
1	PACKET_ID	SEGMENT_ID	PACKET_TYPE	CONTROL_PACKET_TYPE/APP_NAME	SOURCE_ID	DESTINATION_ID	TRANSMITTER_ID	RECEIVER_ID	SOURCE_IP	DESTINATION_IP	GATEWAY_IP	NEXT_HOP_IP
47	0	N/A	Control_Packet	ARP_Request	NODE-1	Broadcast-0	NODE-1	ACCESSPOINT-4	192.169.0.2	255.255.255.255	-	-
58	1	0	CBR	Attack_Information_Flow	NODE-1	NODE-9	NODE-1	ACCESSPOINT-4	192.169.0.2	192.169.0.1	192.169.0.3	192.169.0.1
61	2	0	CBR	Attack_Information_Flow	NODE-1	NODE-9	NODE-1	ACCESSPOINT-4	192.169.0.2	192.169.0.1	192.169.0.3	192.169.0.1
67	3	0	CBR	Attack_Information_Flow	NODE-1	NODE-9	NODE-1	ACCESSPOINT-4	192.169.0.2	192.169.0.1	192.169.0.3	192.169.0.1
73	4	0	CBR	Attack_Information_Flow	NODE-1	NODE-9	NODE-1	ACCESSPOINT-4	192.169.0.2	192.169.0.1	192.169.0.3	192.169.0.1



Additionally, ARP logs generated for each node can be accessed from simulation results window under logs. Each log file contains ARP table entries, including the IP-to-MAC address mappings and the type of each entry in the table.





	A	B	C	D
1	IP_ADDRESS	MAC_ADDRESS	TYPE	
2	TIME(micro sec) 761.090000			
3	255.255.255.255	FFFFFFFFFFFF	STATIC	
4	192.169.0.1	155D00007001	STATIC	
5	11.0.0.3	155D00007002	STATIC	
6	192.169.0.2	155D00003001	DYNAMIC	
7				