Task1: Create a library of probe stuffing code.
Status: Complete

A. Library functions available to users:

1. void set_wifi_interface (char* interface)
2. void set_effort (int delivery_effort)
3. void set_probe_tx_interval (double transmission_interval)
4. void set_data_to_stuff_size_limit (int stuff_data_size_limit)
5. int start_probe_stuffing (_Bool data_input_mode, char* data_source)

Details:

1. void set_wifi_interface (char* interface)

Sets the network interface to be used for transmitting a probe request
Parameter: interface - name of the network interface

2. void set_effort (int delivery_effort)

Parameter: delivery_effort
delivery_effort = 1 for Best delivery effort
delivery_effort = 2 for Guaranteed effort.
Default value = 2

3. void set_probe_tx_interval (double transmission_interval)

Sets the time in seconds between the generation of 2 consecutive probe requests.
Default value = 1.0 i.e., a probe request is generated after every 1 second
Note: More than 1 probe request is generated when the size of data to be transmitted is more than the max size of data that can be stuffed in a single probe request.

4. void set_data_to_stuff_size_limit (int stuff_data_size_limit)

Sets the size limit in bytes for the data to be stuffed in 1 probe request.
Parameter: stuff_data_size_limit
Max size = 1394 bytes
Default = 300 bytes

Example: if stuff_data_size_limit=2, then 2 bytes of data is stuffed in 1 probe request

5. int start_probe_stuffing (_Bool data_input_mode, char* data_source)

This is the main function to start the transmission.
Parameter:
data_input_mode = 0 to read data from the user
data_input_mode = 1 to read data from a file
When data_input_mode = 0, data_source is the data to be stuffed
When data_input_mode = 1, data_source is the name of the file from which data is to be read

Example:
start_probe_stuffing (0, "Hello_World"); //Hello_World is the data
start_probe_stuffing (1, "file.txt"); //file.txt is the name of the file that contains data

B. How to create probeStuffing shared library :

1. libnl (netlink) header files are not installed in standard C include path. So first set the libnl (netlink) library header files path using the following command:

export C_INCLUDE_PATH=/usr/include/libnl3:$C_INCLUDE_PATH

2. Probe stuffing code contains four source files, namely, **utils.c**, **nl_callbacks.c**, **probe_stuffing.c** and **start_probe_stuffing.c**
Compile the above mentioned source file to a position-independent object files using -fPIC option using the following command:

gcc -c -fPIC utils.c
gcc -c -fPIC nl_callbacks.c
gcc -c -fPIC probe_stuffing.c
gcc -c -fPIC start_probe_stuffing.c

3. Combine the object files into a shared library and explicitly specifying it depends on nl-3 and nl-genl-3 libraries using the following command:

gcc -shared -fPIC -o libprobeStuffing.so utils.o nl_callbacks.o probe_stuffing.o start_probe_stuffing.o -lnl-genl-3 -lnl-3

It creates a shared library **libprobeStuffing.so** in the current working directory.

C. How to use probeStuffing library in a program:
1. To use the probe_stuffing library in your program, include library header in your program.
Example:
#include<headers.h>

where headers.h is the name of the library header file.

2. Necessary function to call:
set_wifi_interface(<Name of the interface>);
start_probe_stuffing(<data_input_mode>, <data_source>);

Optional functions to set different configurations:
set_effort(<effort 1 or 2>);
set_probe_tx_interval(<time in seconds>);
set_data_to_stuff_size_limit(<size in bytes>);

Example Program: MyApplication.c

```
#include<headers.h>

int main () {

        set_wifi_interface("wlp19s0");
        start_probe_stuffing (0, "Hello_World");

        return 0;
}
```

To compile the program:

```
gcc -o MyApplication MyApplication.c -I. -L. -lprobeStuffing -Wl, -rpath=.
```

Note: -I. flag tells the compiler to find the library header file in the current directory for linking
-L. flag tells the compiler to find the library in the current directory for linking
-l flag specify the name of the probe stuffing library

Now we don't need to link the netlink library again with our executable.

To run the executable:

```
sudo ./MyApplication
```

Note:
When you link a program with a shared library, the linker does not put the full path to the shared library in the resulting executable. Instead, it places only the name of the shared library. When the program is actually run, the system searches for the shared library and loads it. The system searches only /lib and /usr/lib, by default. If a shared library that is linked into your program is installed outside those directories, it will not be found, and the system will refuse to run the program. There are 3 solutions to this problem:

1. Placing the probeStuffing library in the aforementioned default locations. This option may require administrator privileges.

2. Set the environment variable LD_LIBRARY_PATH to the directory containing the shared libraries, which is current directory (".") in most of the cases using the following command:
export LD_LIBRARY_PATH=.:$ LD_LIBRARY_PATH
library path is specified after the = sign. Here it is "." which is the current directory.
This option is usually not recommended because asking your users to set **LD_LIBRARY_PATH** means an extra step for them. Because each user has to do this individually, this is a substantial additional burden.

3. Use the -Wl, -rpath=DIR option when linking the program with shared libraries that aren't located in standard system library directories. This will write some information into the executable to tell the loader that it should search for required shared libraries in DIR before it tries the default places.

Task 2: Eliminate the need for sudo permission every time to run the application
Status: Complete

Command: **sudo setcap cap_net_admin+ep ./MyApplication**
This command allows non_root users to run the applications that are using probe stuffing library.

Now to run the application:
./MyApplication

Reason:
Probe stuffing code uses libnl (netlink) library for communication between Userspace and kernel space. Netlink always checks a flag CAP_NET_ADMIN to ensure whether an application is allowed to access sockets or not. Currently netlink socket permissions are controlled by the CAP_NET_ADMIN capability of the client. By default, only root users have permissions.

CAP_NET_ADMIN - Allow various network-related operations (socket programming, multicasting, Access to network interfaces etc.)
Linux capabilities ensure that only someone with CAP_NET_ADMIN capability (typically, the root user) is allowed to open sockets. So that is why we use setcap command so that non root user application has the capability to access sockets.

Task 3: Avoid the need to root android phone to run the application

The following three commands are required to perform passive/active scanning:
1. NL80211_CMD_TRIGGER_SCAN – the command for starting a new IEEE 802.11 scan (active/passive)
2. NL80211_CMD_NEW_SCAN_RESULTS – returns status of scan (success or failure)
3. NL80211_CMD_GET_SCAN - the command for getting the scan results

1 and 2 need root privileges. In the thesis **Flexible Information Broadcasting using Beacon Stuffing in IEEE 802.11 Networks,** they used Android API function ***startScan()*** in their java application instead of using command 1 and 2 and then called their native binary function (command 3) to forward the results to the java application.

In other words, they replaced command 1 and 2 with Android API’s startScan() to bypass root privileges. This startScan() function can only perform passive scanning. Therefore, we can’t use this approach for probe stuffing library. As for now, there is no other alternative for these two commands.