# CS111 - Project 4: Internet of Things Security

## INTRODUCTION:

The new Internet of Things (IoT) thrust is one of the most important trends in technology today. IoT systems provide sensors and actuators in our global environment, on the scale of nations and cities, to individual buildings, homes, and vehicles. In this project you will bring up an embedded sensor simulation on an new embedded platform (Intel Edison), demonstrate the hack-ability of an embedded wireless device, and then secure that communication with the Transport Layer Security protocol (successor to SSL).

## RELATION TO READING AND LECTURES:

This lab builds on the socket based communication you prototyped in Project 1, and the distributed system security discussion presented in Saltzer chapter 11 and Lecture 16.

## PROJECT OBJECTIVES:

- demonstrate the ability to bring up an embedded system.
- demonstrate the ability to develop, download, and run, and debug software on an embedded system.
- demonstrate the hackability of unsecured wireless communications with an embedded device.
- gain familiarity with the Transport Layer Security protocol.
- demonstrate the ability to secure a pre-existing protocol.

## DELIVERABLES:

A single tarball (.tar.gz) containing:

- a single C source module that compiles cleanly (with no errors or warnings).
- a Makefile to build the program and the tarball.
- 4 screenshots for the UDP client
- 4 screenshots for the TLS client
- a README file describing each of the included files, the answers to the required questions, and any other information about your submission that you would like to bring to our attention (e.g. limitations, features, testing methodology, use of slip days).

## REQUIRED HARDWARE:

- (1) Intel Eddison board ... which you will check out from the TA and which you must return at the end of the lab (or receive a zero).
- (2) micro-USB cables
- You will also need access to a server that is running tshark.

## BACKGROUND:

IoT systems also include the rapidly expanding world of wearable devices for the field of Wireless Health or Mobile Health (mHealth). A critical requirement of Wireless Health is the support of patient monitoring by applying remote sensors. The sensors monitor such vital signs as heart rate, pulse, blood sugar levels, and many other critical health indicators. Generally, while the devices performing the sensing are located on or near the

patient's body, the data ultimately must go to a health care facility for analysis. The first step in that process is using a wireless network to move the data gathered from the patient to some other nearby machine, which typically in turn passes it over the Internet to a doctor, hospital, or other health care facility. So the sensor must not only be able to observe the patient's condition, it must also operate as a wireless host computer. In many cases, the sensor is configurable, perhaps monitoring different vital signs, or monitoring them at different frequencies, or at different levels of sensitivity. In such cases, not only must the sensor's communications and computational capabilities move data off the system, but they must accept commands controlling their behavior coming in over the wireless network.

Health data is inherently private, yet wireless networking is inherently open. Anyone who puts up an antenna within range of a wireless device will hear its signal. So unless measures were taken to protect the wireless communications from sensor to base station, eavesdroppers could learn all the information they exchange. Such a vulnerability is extremely undesirable. As discussed in the reading and lectures, the fundamental way to address this kind of vulnerability is to encrypt the data, making it incomprehensible to anyone except the sender (the Edison here) and the receiver (the base station that will pass the data to the health care facility).

These devices are now based on compact platforms including the Intel Edison IoT platform that you will be provided. This platform uses the Linux Operating System as its software base. Fortunately, the Linux system contains high quality built-in mechanisms to perform encryption and decryption. One only has to use them properly (which can be challenging, itself). You may find this TLS tutorial to be helpful.

# PROJECT DESCRIPTION:

You will check out an Edison (which must be returned at the end of the project), and bring up a heart rate monitor simulation that communicates with a server that we will provide. You will run this software and use the tshark network monitoring tool to observe the client/server communication, and demonstrate the ability to hack this communication. Finally, you will extend the application to secure client/server communication using the TLS protocol, and demonstrate both the privacy and the hack-resistance of your solution.

## PART A: Bring-up

1. Obtain and Edison from your TA, plug it in, and bring it up. If you are new to working with the Intel Edison boards, please see the Edison IoT Tutorial
2. Log in to your Edison using the SSH protocol. These instructions are based on using MobaXterm for SFTP and SSH, but many other tools will work.
3. Download the project starter code.
4. Transfer the downloaded files to your Edison (e.g. drag and drop using MobaXterm).
5. (on the Edison) go to the heartbeat/UDP/client directory and run the command: make.
6. (from your lab TA) obtain host and port information for:

   - the UDP test server
   - the TLS test server
   - the tshark UDP server

- Add this information to the config_file in each subdirectory under the client directory.

## PART B: Demonstration of Insecurity

1. Start the UDP client on the edison.
   a. (on the Edison) go to the heart_rate/UDP/client directory, and run the command:

./udp_client.

2. Keep this command running for the full duration of part B.

    a. Take a screen-shot of the first 10 lines of output, and save it as Lab4-B-1.jpg (which you will submit).
    b. Examine the output that is being sent to the server, and write a brief description of that output and the security/privacy implications of what you see. Put these comments in your README file under the heading **Lab4-B-1**.

1. Packet-sniff the traffic between Edison and the server:
    a. open a new terminal window to the Edison.
    b. go to the tshark_UDP sub-directory, and run the command

./get_tshark *port#*

(where *port#* is the port on which the UDP server is running).

    a. Take a screen-shot of this output, and save it as as Lab4-B-2.jpg (which you will submit).
    b. Examine the output from the packet sniffer, and write a brief description of security/privacy implications of what you see. Put these comments in your README file under the heading **Lab4-B-2**.
    c. Close the tshark_UDP window.

1. Change the simulated heart-beat and examine the traffic.
    a. open a new terminal window to the Edison.
    b. go to the heart_rate/UDP/client sub-directory, and run the command

./set_rate <0-5>

    a. Take a screen-shot of udp_client output, and save it as as Lab4-B-3.jpg (which you will submit).
    b. Write a brief description of how the output changes. Put these comments in your README file under the heading **Lab4-B-3**.
    c. DO NOT CLOSE THIS TERMINAL WINDOW YET.

1. Start an attack on the heart-rate monitor
    a. In the same terminal window (used for step 3) run the command

./start_attack

    a. Take a screen-shot of udp_client output, and save it as as Lab4-B-4.jpg (which you will submit).
    b. Examine the output, and write a brief description of how the output changes. Put these comments in your README file under the heading **Lab4-B-4**.

1. Shut down the test by typing ^C (control-C) in the udp_client window and closing all of the terminal windows.

## PART C: Securing the Communications Chanel

This set of tasks involves improving the UDP version of the client code to provide better security, using TLS. After making the required changes, you will run your new client code and examine the output from a packet analysis tool installed at the server location. You will use this output to understand why encrypted communication channels are crucial.

In the TLS/client subdirectory of the downloaded project materials you will find a file tls_client.c. In the TLS/client subdirectory of the downloaded project materials you will find a file tls_client.c. Navigate to the folder "../heart_beat/TLS/client/" and type "make"

## PART D: Demonstration of Improved Security

1. Start your new TLS client on the Edison.
    a. execute your new TLS client by running the command

./tls_client.

2. Keep this command running for the full duration of part D.

    a. Take a screen-shot of the first 10 lines of output, and save it as Lab4-D-1.jpg (which you will submit).
    b. Examine the output that is being sent to the server, and write a brief description of that output. Put these comments in your README file under the heading **Lab4-D-1**.

1. Packet-sniff the traffic between Edison and the server:
    a. open a new terminal window to the Edison.
    b. go to the tshark_UDP sub-directory, and run the command

./get_tshark *port#*

(where *port#* is the port on which the TLS server is running).

    a. Take a screen-shot of this output, and save it as as Lab4-D-2.jpg (which you will submit).
    b. Examine the output from the packet sniffer, and write a brief description how this is different from the output you saw in part B. Put these comments in your README file under the heading **Lab4-D-2**.
    c. Close the tshark_UDP window.

1. Change the simulated heart-beat and examine the traffic.
    a. open a new terminal window to the Edison.
    b. go to the heart_rate/UDP/client sub-directory, and run the command

./set_rate <0-5>

    a. Take a screen-shot of tls_client output, and save it as as Lab4-D-3.jpg (which you will submit).
    b. Write a brief description of how the output changes. Put these comments in your README file under the heading **Lab4-D-3**.
    c. DO NOT CLOSE THIS TERMINAL WINDOW YET.

1. Start an attack on the heart-rate monitor
    a. In the same terminal window (used for step 3) run the command

./start_attack

    a. Take a screen-shot of tls_client output, and save it as as Lab4-D-4.jpg (which you will submit).
    b. Examine the output, and write a brief description of how the output changes. Put these comments in your README file under the heading **Lab4-D-4**.

1. Shut down the test by typing ^C (control-C) in the tls_client window and closing all of the terminal windows.

**PART E: Demonstration of Improved Security and Resolution of Errors**

This set of tasks involves developing a new TLS client that resolves the condition leading to the behavior associated with unmatched message response noted in step 6 above. You may use the tls_client.c program to create a new program.

First, it is important to note that tls_client sends a message to the server via the SSL_write() function. It expects a matching reply via SSL_read().

However, if an unexpected reply message is sent by the server, then the return of SSL_read() will not match that

of the message sent by SSL_write(). The behavior appearing in Step 7 will result. This unexpected behavior could result from multiple forms of system error or system compromise.

Your task is to resolve this behavior through two steps:

1. Extend the tls_client.c program to include a second thread of execution. This thread will include SSL_read() actions.
2. Extend the tls_client.c program to also include logging of messages written to a log file.

After making the required changes, you will

1. Execute your new client code and examine the output of the client.
2. During execution of the client, execute the command "./set_rate 1"
3. Comment on how your "./tls_client" reacts and take a screenshot of the output that appears both before and after the execution, "./set_rate 1" and save it as **Lab4-E-1.jpg**
4. Comment on how your dual thread tls_client avoids errors that are discussed above.
5. Examine the log file and save the log file contests that appears both before and after the execution, "./set_rate 1" and save it as **Lab4-E-2.txt**

# RUBRIC:

Points for this project will be awarded:

| Value | Feature |
|---|---|
| | **packaging and build** |
| 3% | untars expected contents |
| 3% | clean build w/default action |
| 1% | correct `make dist` |
| 3% | reasonablness of `README contents` |
| | **Functionality** |
| 10% | Correctly Encrypted Client/Server Communication |
| | **UDP client screen shots** |
| 5% | Lab4-B-1.jpg and description |
| 10% | Lab4-B-2.jpg and description |
| 5% | Lab4-B-3.jpg and description |
| 10% | Lab4-B-4.jpg and description |
| | **TLS client screen shots** |
| 5% | Lab4-D-1.jpg and description |
| 10% | Lab4-D-2.jpg |
| 5% | Lab4-D-2 description |
| 5% | Lab4-D-3.jpg and description |
| 10% | Lab4-D-4.jpg |
| 5% | Lab4-D-4 description |
| | **code/package review** |
| 5% | correctness/reasonableness of TLS implementation |
| 5% | overall clarity |

# SUBMISSION:

Project 4 is due before midnight on Monday June 6, 2016. Slip-days can be used on this project, but if it is turned in after June 8, you may have to take an incomplete in the class, which we will change as soon as we are able to finish the grading.

Your tarball should have a name of the form `lab4-`*studentID*`.tar.gz`, and should be submitted via CCLE.

We will test your work on a SEASnet GNU/Linux server running RHEL 7 (this is on lnxsrv09). You would be well advised to test your submission on that platform before submitting it.