Securing IoT Infrastructure

UNIVERSITY OF SALERNO
Department of Computer Science
Master of Science in IOT
IoT Security

Author:

Blessings Nthara

Muhammad Yasir

Mohamed Rammszy Careem

Supervisor:

Prof. Esposito Christian

INTRODUCTION:

- IoT is an advancing technology that could greatly transform multiple industries and everyday life.
- IoT is a system of connected physical devices.
- IoT gathers and shares data through the internet.
- IoT automates tasks and supplies valuable data for analysis and decision-making.
- In today's interconnected world, data security is of utmost importance.
- Different security protocols are essential for safeguarding sensitive information and ensuring secure communication.

PURPOSE STATEMENT:

- Safeguard sensitive information and ensure secure communication.
- Protect valuable data and mitigate potential threats.
- Establish a secure environment for the collection, exchange, and analysis of data in IoT networks.
- Maintain the trust and integrity of IoT systems while minimizing the risk of unauthorized access or data breaches.
- Address the evolving challenges and vulnerabilities associated with the rapid growth of IoT technology.

EXPERIMENT:

- Arduino board equipped with an ultrasonic sensor to collect data, which is then encrypted and decrypted for secure transmission.
- NodeMCU board with a raindrop sensor to capture data, which is subsequently transmitted using an API Key.
- NodeMCU board along with a DHT11 sensor to gather data, which is then sent to an MQTT broker using TLS/SSL certificate.

OBJECTIVE:

- The distance of an object is measured by the HC-SR04 sensor in the Arduino Uno Rev 2. The Arduino Rev Uno 2 is connected to WiFi and sends the encrypted data to an Apache server. The server decrypts the data using PHP and OpenSL and sends it back.

- The raindrop level is detected by the MH-RD sensor in the NodeMCU. The NodeMCU is connected to WiFi and transmits the data to ThingSpeak using an API Key for security.

- The temperature and humidity are measured by the DHT11 sensor in the NodeMCU. The NodeMCU is connected to WiFi and securely communicates the data to HiveMQTT using TLS/SSL certificate and user authentication.

TECH STACK:

Operating System:

- Ubuntu Linux: An open-source operating system based on the Linux kernel known for its stability and security.

Software:

- Arduino Integrated Development Environment (IDE): A software platform used to write, compile, and upload code to Arduino boards. It provides a user-friendly interface for developing Arduino projects.

- PHP: A popular server-side scripting language used for web development. It is often used in conjunction with Apache to create dynamic web applications.

- Apache: A widely used open-source web server software that allows you to host websites and serve web content. It works seamlessly with PHP and is a common choice for hosting PHP applications.

- HiveMQ: HiveMQ is a scalable MQTT (Message Queuing Telemetry Transport) broker that enables communication between devices in an Internet of Things (IoT) ecosystem. It facilitates the exchange of messages between devices using the publish-subscribe pattern.

- ThingSpeak: ThingSpeak is an IoT platform that allows you to collect, analyze, and visualize data from connected devices. It provides APIs and tools to store and retrieve data, as well as create real-time dashboards and charts.

Hardware:

- Arduino Uno Rev 2: Arduino is an open-source electronics platform that provides microcontrollers for building digital devices and interactive objects. The Arduino Rev 2 refers to a specific version of the Arduino board.

- NodeMCU ESP8266: The NodeMCU ESP8266 is a low-cost Wi-Fi microcontroller board based on the ESP8266 chip. It is commonly used for IoT projects and can be programmed using the Arduino IDE.

- Ultrasonic Range Finder Sensor (HC SR04): An ultrasonic sensor used to measure distances. It emits ultrasonic waves and measures the time it takes for the waves to bounce back, allowing you to calculate the distance to an object.

- DHT11 Temperature and Humidity Sensor: The DHT11 is a basic digital sensor that can measure temperature and humidity in the surrounding environment. It provides accurate readings for basic weather monitoring or climate control applications.

- MH-RD Raindrop Sensor: The MH-RD Raindrop Sensor is a sensor that can detect rain or the presence of water. It is commonly used in weather stations or irrigation systems to detect rainfall.

IMPLEMENTATION - ARDUINO BOARD EQUIPPED WITH AN ULTRASONIC SENSOR TO COLLECT DATA, WHICH IS THEN ENCRYPTED AND DECRYPTED FOR SECURE TRANSMISSION:

1. It includes several libraries:

- `SPI.h`: Provides functions for communicating with devices using the Serial Peripheral Interface (SPI) protocol. It is often used to communicate with external devices such as sensors, displays, and memory chips.

- `WiFiNINA.h`: Designed for Arduino boards that use the WiFiNINA module, which provides Wi-Fi connectivity. It includes functions for connecting to Wi-Fi networks, performing HTTP requests, and managing network connections.

- `AES.h`, `AESLib.h`, and `AES_config.h`: Provide implementations of the Advanced Encryption Standard (AES) algorithm. AES is a widely used encryption standard, and these libraries allow you to encrypt and decrypt data using AES.

- `Base64.h`: Provides functions for encoding and decoding data in Base64 format. Base64 is a binary-to-text encoding scheme commonly used for transmitting binary data over text-based protocols such as HTTP or email.

2. It defines global variables and constants needed for the sketch:
  - `cipher[1000]` and `b64[1000]`: Arrays to store the encrypted data and base64-encoded data.
  - `trigPin` and `echoPin`: Pin numbers for a distance measurement sensor (ultrasonic sensor).
  - `duration` and `distance`: Variables to store the measured duration and calculated distance from the sensor.
  - `ssid` and `pass`: Wi-Fi network credentials (SSID and password).
  - `keyIndex`: Index for the encryption key.
  - `server`: IP address of the server.
  - `client`: An instance of the `WiFiClient` class used for communication with the server.

3. The `do_encrypt` function is defined. It takes three parameters:
   - `msg`: The message to be encrypted (distance value converted to a string).
   - `key_str`: The secret key used for encryption (16-byte string).
   - `iv_str`: The initialization vector used for encryption (16-byte string).

   Inside the function:
   - An initialization vector (`iv`) is created by copying the content of `iv_str` into a byte array.
   - The message is base64 encoded and stored in `b64`.
   - The size of the encrypted data is calculated, and padding is applied if needed.
   - An array (`plain_p`) is created to hold the padded message.
   - AES-128-CBC encryption is performed using the provided key and initialization vector.
   - The encrypted data is base64 encoded and stored in `b64`.
   - The encrypted data is printed to the serial monitor.
   - A connection is made to the server using the `client` object.
   - An HTTP GET request is sent to the server with the encrypted data as a parameter.

4. The `setup` function is called once when the Arduino board is powered on or reset:
   - Serial communication is initialized with a baud rate of 9600.
   - The sketch waits for the serial port to connect.
   - The status of the Wi-Fi module is checked, and if there is an issue, an error message is printed.
   - The firmware version of the Wi-Fi module is checked for compatibility.
   - The sketch attempts to connect to the specified Wi-Fi network using the provided credentials.
   - If the connection is successful, the Wi-Fi status is printed.
   - The `printWifiStatus` function is called to display additional Wi-Fi information.
   - A connection to the server is initiated (commented out in the code).

5. The `loop` function is called repeatedly after the `setup` function:
   - It checks for incoming bytes from the server and prints them to the serial monitor.
   - It measures the distance using the ultrasonic sensor and calculates the distance in centimeters.
   - The distance value is converted to a string (`msg`).
   - The `do_encrypt` function is called with the message, encryption key, and initialization vector.
   - A delay of 5 seconds is applied.
   - If the server is disconnected, the client is stopped, and the sketch enters an infinite loop.

6. The `printWifiStatus` function displays the current Wi-Fi status, including the SSID, IP address, and signal strength (RSSI).

CODE - ARDUINO BOARD EQUIPPED WITH AN ULTRASONIC SENSOR TO COLLECT DATA, WHICH IS THEN ENCRYPTED AND DECRYPTED FOR SECURE TRANSMISSION:

AES_Encrypt_UltrasonicSensor_ArduinoUnoRev2.ino

```
#include <SPI.h>
#include <WiFiNINA.h>
#include <AES.h>

#include <AESLib.h>

#include <AES_config.h>

#include <Base64.h>

AES aes;

byte cipher[1000];

char b64[1000];


const int trigPin = 9;
const int echoPin = 10;

long duration;
int distance;


char ssid[] = "****";
char pass[] = "****";
int keyIndex = 0;

int status = WL_IDLE_STATUS;



char server[] = "192.168.145.218";

```

CODE - DECRYPTION

```php
<?php

$data = isset($_GET["enc"]);

echo file_put_contents("test.txt",$_GET["enc"]);

$replaced = str_replace(' ', '+', $_GET["enc"]);


$raw = $replaced;

// Convert the encrypted data from base64 to binary
$encryptedData = $raw;

// Define the encryption key and IV
$key = "aaaaaaaaaaaaaaaa"; // Encryption key (16 characters for AES-128)
$iv = "aaaaaaaaaaaaaaaa"; // Initialization Vector (IV)

// Decrypt the data
$decryptedData = openssl_decrypt($encryptedData, 'aes-128-cbc', $key, OPENSSL_ZERO_PADDING, $iv);

// Output the decrypted data
echo "Decrypted Data: " . base64_decode($decryptedData);

?>
```

SECURITY FEATURES IMPLEMENTED:

- **AES Encryption:** The script uses AES (Advanced Encryption Standard) with a 128-bit key size and CBC (Cipher Block Chaining) mode for encryption. AES is a widely adopted symmetric encryption algorithm, known for its security and efficiency. By encrypting the data with AES, the script ensures that the transmitted distance values are protected from unauthorized access.

- **Initialization Vector (IV):** The script includes an initialization vector (`iv`) that is used in conjunction with the encryption key. The IV adds randomness to the encryption process, making it more secure. It ensures that even if the same message is encrypted multiple times, the resulting cipher text will be different, enhancing the security of the encryption.

- **Key Management:** The script defines a 16-byte secret key (`key_str`) for encryption. While the provided key in the script is a static value, in real-world scenarios, it is important to use a strong, random, and unique key for each encryption operation. Proper key management practices, such as secure key generation, storage, and distribution, are essential for maintaining the security of encrypted data.

- **Base64 Encoding:** The script uses Base64 encoding to represent the encrypted data in a printable and ASCII-safe format. Base64 encoding is not a security measure itself but facilitates the transmission of binary data over systems that expect only printable ASCII characters. It ensures that the encrypted data can be properly transmitted and received without any data loss or corruption.

- **Wi-Fi Security:** The script utilizes Wi-Fi communication to send the encrypted data to a server. The security of the Wi-Fi connection depends on the security protocols supported by the Wi-Fi network. In this case, the script connects to a network using a pre-shared key (WPA/WPA2). It is recommended to use secure Wi-Fi networks with strong encryption protocols (e.g., WPA3) and unique, complex passwords to protect the integrity and confidentiality of the transmitted data.

- **Server-Side Security:** The script sends the encrypted data to a specific server (IP address). The security of the server depends on various factors, including the server's configuration, authentication mechanisms, and encryption protocols (if applicable). It is crucial to ensure that the server is properly secured, with appropriate access controls, secure communication channels (HTTPS), and proper handling of the received encrypted data.

IMPLEMENTATION - NODEMCU BOARD WITH A RAINDROP SENSOR TO CAPTURE DATA, WHICH IS SUBSEQUENTLY TRANSMITTED USING AN API KEY + DHT11 SENSOR TO GATHER DATA, WHICH IS THEN SENT TO AN MQTT BROKER USING TLS/SSL CERTIFICATE:

1. Libraries:
- `ESP8266WiFi.h`: Provides the necessary functions to connect to a Wi-Fi network.
- `PubSubClient.h`: Enables MQTT communication with an MQTT broker.
- `DHT.h`: Allows reading data from DHT temperature and humidity sensors.
- `time.h` and `TZ.h`: Provides functions for time and time zone management.
- `FS.h` and `LittleFS.h`: Allow access to the filesystem on the ESP8266.
- `CertStoreBearSSL.h`: Enables secure connections using SSL/TLS certificates.
- `ThingSpeak.h`: Provides functions to upload data to the ThingSpeak IoT platform.

2. Wi-Fi and MQTT Configuration:
- `ssid`: The name of the Wi-Fi network to connect to.
- `password`: The password for the Wi-Fi network.
- `mqtt_server`: The MQTT broker server address.

3. Temperature Sensor Configuration:
- `DHT_PIN`: The pin number to which the sensor is connected.
- `DHT_TYPE`: The type of DHT sensor (in this case, DHT11).

4. Raindrop Sensor Configuration:
- `raindropPin`: The analog input pin number to which the raindrop sensor is connected.

5. Other Variables and Objects:
- `espClient`: An instance of the `WiFiClientSecure` class used for secure connections.
- `client`: A pointer to the `PubSubClient` object used for MQTT communication.
- `lastMsg`: A variable to track the time of the last message sent.
- `msg`: A character array to store the MQTT message.
- `value`: A variable used to increment the message value.
- `temperature`: A variable to store the temperature value read from the DHT sensor.
- `dht`: An instance of the `DHT` class used for reading data from the DHT sensor.
- `wifiClient`: An instance of the `WiFiClient` class used for ThingSpeak communication.
- `server`: The ThingSpeak API server address.
- `apiKey`: The ThingSpeak API key for accessing the channel.
- `channelID`: The ID of the ThingSpeak channel to which the data will be uploaded.

6. Function Definitions:
- `setup_wifi()`: Connects to the specified Wi-Fi network.
- `setDateTime()`: Sets the date and time using NTP (Network Time Protocol).
- `callback()`: A callback function that is triggered when a message is received on the subscribed topic.
- `reconnect()`: Reconnects to the MQTT broker if the connection is lost.

7. setup():
The `setup()` function is called once when the device is powered on or reset. It initializes the serial communication, sets up the Wi-Fi connection, sets the date and time, initializes the LED pin, initializes the certificate store, creates a secure Wi-Fi client, creates an MQTT client, sets the MQTT server and callback, and begins the ThingSpeak communication.

8. loop():
The `loop()` function is the main function that runs repeatedly after the `setup()` function. It checks if the MQTT client is connected and calls the `client->loop()` function to maintain the MQTT connection. It also checks if enough time has passed since the last message was sent and performs the following actions:

- Reads the temperature and humidity from the DHT sensor.
- Checks if the sensor readings are valid.
- Formats the sensor data into a message string.
- Publishes the message to the MQTT broker with the topic "temp".
- Reads the value from the raindrop sensor.
- Prints the raindrop value to the serial monitor.
- Sets the value of the first field in the ThingSpeak channel to the raindrop value.
- Writes the data to the ThingSpeak channel using the API key.
- Prints the response from the ThingSpeak API to the serial monitor.
- `loop()` function introduces a delay of 20 seconds before repeating the process.

CODE - NODEMCU BOARD WITH A RAINDROP SENSOR TO CAPTURE DATA, WHICH IS SUBSEQUENTLY TRANSMITTED USING AN API KEY + DHT11 SENSOR TO GATHER DATA, WHICH IS THEN SENT TO AN MQTT BROKER USING TLS/SSL CERTIFICATE:

MQTT_Certificate_APIKey_NodeMCU.ino

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>
#include <time.h>
#include <TZ.h>
#include <FS.h>
#include <LittleFS.h>
#include <CertStoreBearSSL.h>
#include <ThingSpeak.h>


const char* ssid = "****";
const char* password = "****";
const char* mqtt_server = "****.s1.eu.hivemq.cloud";

//Declaration of Pins for Temperature Sensor
const int DHT_PIN = D2;
const int DHT_TYPE = DHT11;

// A single, global CertStore which can be used by all connections.
// Needs to stay live the entire time any of the WiFiClientBearSSLs
// are present.
BearSSL::CertStore certStore;

WiFiClientSecure espClient;
PubSubClient * client;
unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE (500)
char msg[MSG_BUFFER_SIZE];
int value = 0;

float temperature = 0.0;
DHT dht(DHT_PIN, DHT_TYPE);
```

SECURITY FEATURES IMPLEMENTED:

- Wi-Fi Connection Security:
  - The Wi-Fi network connection is established using WPA/WPA2 security by providing the network's SSID and password.
  - The connection is made using the `WiFiClientSecure` class, which provides a secure connection over TLS/SSL.

- MQTT Connection Security:
  - The MQTT broker connection is made using the `PubSubClient` library's `WiFiClientSecure` implementation, which enables secure communication over TLS/SSL.
  - The MQTT broker server is specified with an IP address or domain name and port number (8883) for a secure MQTT connection.
  - The MQTT client uses TLS/SSL certificates stored in the `certStore` object, which ensures the authenticity of the MQTT broker.

- ThingSpeak Communication Security:
  - The script uses the ThingSpeak API to upload data to the ThingSpeak platform.
  - The communication with ThingSpeak is performed over HTTPS, which provides encryption and secure data transmission.
  - The API key is used to authenticate and authorize access to the ThingSpeak channel.

- Secure Time Synchronization:
  - The script uses NTP (Network Time Protocol) to synchronize the device's time with external time servers.
  - By synchronizing the time, the script ensures accurate timestamping and supports secure certificate validation.

- SSL/TLS Certificates:
  - The script utilizes the `CertStoreBearSSL` library to manage and store SSL/TLS certificates securely.
  - The certificate store is initialized with certificates stored in the LittleFS filesystem.
  - The MQTT client and Wi-Fi client use the certificate store to verify the authenticity of the MQTT broker and establish secure connections.

TLS/SSL CERTIFICATE CREATION:

- Download the repository: Obtain the necessary files, including the "certs-from-mozilla.py" script.
- Execute the script: Open the "certs-from-mozilla.py" script in a Python IDE and run it to generate the "certs.ar" file.
- Organize Arduino IDE: Access the Sketch folder by selecting Sketch>Show Sketch Folder in Arduino IDE.
- Create a data folder: Inside the Sketch folder, create a new folder named "data."
- Move generated file: Place the "certs.ar" file into the "data" folder.
- Adjust flash size (if needed): In the Arduino IDE's Tools menu, select an appropriate flash size based on file size.
- Upload files: Use Tools>ESP8266 LittleFS Data Upload to upload files from the "data" folder to the board's flash memory.
- Wait for completion: Allow a few seconds for the upload process to finish.

SECURITY ASPECTS ALTERNATIVES:

- **Secure Boot:** Implement a secure boot process that verifies the integrity and authenticity of firmware and software components during the device boot-up process. This ensures that only trusted and unaltered software is executed.

- **Firmware Updates and Patch Management:** Establish a robust and secure mechanism for distributing firmware updates and patches to IoT devices. This includes securely delivering updates, verifying their authenticity, and applying them in a timely manner to address security vulnerabilities and bugs.

- **Intrusion Detection and Monitoring:** Deploy intrusion detection systems (IDS) or intrusion prevention systems (IPS) to detect and respond to any suspicious activities or unauthorized access attempts. Additionally, implement robust logging and monitoring mechanisms to track and analyze security events.


THANK YOU