Keycard-Power Relay Access System

Zachary Hancock
Grand Valley State University

Keycard Reader - Power Relay Access System

By
Zachary Hancock

A project submitted in partial fulfillment of the requirements for the degree of

Master of Science in

Computer Information Systems

at

Grand Valley State University

April, 2020

Dr. Vijay Bhuse **04/29/2020**

Table of Contents

Abstract

At GVSU, we have various pieces of high-tech equipment and tools that need access to be monitored by each individual piece of equipment. Each lab in the sciences is protected by a keycard access door. If you have access to that room, you have access to everything in that room. This is not ideal in a couple of rooms managed by the CLAS Department. Within these rooms, we have many power tools that require safety training, as well as many expensive microscopes that need training. This need gave us an idea to reduce access to instruments by training.

What we came up with was an affordable keycard reader-power relay system. This system has three parts to it: a keycard reader module, an IoT power relay, and a Raspberry Pi. The three devices work together to lock down a device and provide access to only those who are allowed to use it, through their CLAS provided keycard.

This system uses the Pi as a Wireless Access Point, a database manager, and a MQTT Broker. We have the keycard reader wait for a keycard to be inserted. Once the keycard is inserted, the reader sends a message off to the Pi. The Pi receives that message, and checks the ID against the database to allow access. It then sends off a message to the power relay to turn on the device. The Pi also logs whenever a keycard is inserted, in order to keep a usage log.

This system will allow the CLAS department to lock down devices, in order to promote proper use and care of the devices. It also gives them a more affordable option to device security compared to the alternatives on the market.

Introduction

For my project, I decided to work on a keycard reader – power relay access system. This idea came about from the CLAS department's need to add additional security and safety to their equipment. The department has high powered equipment, such as power tools, saws, drills, and also high expense equipment like confocal and UV microscopes. By being able to control access, we can ensure that our students are able to work with equipment safely, correctly and efficiently. The system allows access at the device level, letting users use the equipment if they are authorized to do so. The system also creates logs of use for each device in case of any issues that need to be corrected, as well as to give us an idea of which devices are worth repairing/upgrading or not.

Background and Related Work

This project is meant to complement our existing security measures already in place. Currently, we have keycard access on every lab door across the sciences. However, this doesn't prevent users from using the equipment inside of each room. With a keycard system already in place, we knew that we wanted to continue to use the existing keycards that students and faculty have. This meant forming some sort of keycard reading system that would turn on each device. The current keycard readers that we have are expensive to implement – about $1500 per door, and requires the use of their system. While using their system isn't too much of a hindrance (we use it already for door access), the expense per device was worth more than the device at times. This pushed us towards a more affordable and scalable solution for the various equipment. Another con of the current system is that each of our current key card readers require a physical network connection. Given that rooms generally have a set amount of network ports, and equipment location can be ever-changing, we decided that the new system had to be wireless.

Program Requirements

The solution that was proposed is a low-cost keycard reader that turns a power relay on and off. The device is plugged into that power relay, and only receives power if the keycard being used is in the database for that device. In order to create the system, we needed a few things:

- A keycard reader module (MiFare RC522 module)
- An IoT Power Relay
- Two NodeMCU mini computers

A user will approach a device, and if they'd like to use it, they'd insert their keycard into the reader enclosure, and leave it until they are finished. The reader will read the card, and communicate with a Raspberry Pi that acts as the Master Control Unit for the room. The Pi determines if the requested device can turn on, or should remain off. If it tells it to turn on, the relay sends power to the outlet that the device is connected to, and the user can then operate the device.

Implementation

The first thing that was necessary for this system was to configure a Raspberry Pi to be the central control unit. In order to do this, the Pi needed to have a few systems implemented. First, we needed a way for the Pi to communicate with the reader systems that were to be implemented, and it needed to be done wirelessly. In order to accomplish that, I turned the Pi into a WAP using hostapd and dnsmasq. I then configured dhcpd to create a wireless network on wlan0. Finally, I needed to configure the hostapd to create the actual network and rules. Given the scope of the test, I only allowed for 10 connections, and made the network hidden.

The next step was to add a communication system for the reader, pi and relay to exchange messages. I decided to use MQTT as the communication standard, since it is already heavily used in IoT devices. I installed Mosquitto MQTT on the Pi, so that it could act as an MQTT broker, to be able to send and receive messages to the readers and relays in production.

Finally, the Pi needed a database system that could communicate with our current Filemaker database, and that could be queried by script in order to check for allowed access. I settled on using mySQL, since that was one of the databases that Filemaker supported as well as was available on Raspbian OS. The plan was to use ODBC in order to update the database on the Pi with data from our Filemaker implementation. The mySQL database on the Pi has two tables, a table for keycard IDs and equipment access, and then a table for keeping usage logs.

The Pi runs a python script to constantly listen for incoming MQTT messages. When an MQTT message is received, it parses out the device name and the keycard ID, and checks for them in the database. If access is allowed, the Pi sends out a message to the relay, telling it to turn on, and a second message telling it when to turn off. If access is not allowed, it does not turn on the device. Also, if access is allowed, but it is outside of the time range when access is allowed, it will not turn on the device.

In order to create this system, it required that one NodeMCU be wired to the reader module, and that the other be wired to the power relay. Once those were wired, we needed to upload code to the NodeMCUs to enable them to do a few things. The reader needed to be able to:

- Recognize a card, and when a card is present/not present
- Send a message with the device name and the keycard ID
- Send a message when a card is removed

Using Arduino, I programmed the reader NodeMCU to connect to the Pi over the wireless network that was created, and connect to the MQTT broker on the Pi as well. It subscribes to messages from the Pi for that reader, and then waits for a keycard. Once a keycard is present, it sends off a MQTT message to the Pi. If access is granted, the light on the keycard reader turns green from red, and will stay green until the card is removed, or the time is outside of the allowed time range. If the card is removed, a message is sent to have the relay turned off.

The relay NodeMCU also needed to be connected to the Pi in order to receive messages. The relay waits for MQTT messages from the Pi to turn on and off. It also has the Time library imported so that it can keep track of the current time, as well as the time that it will need to shut off due to time limits.

Results, Evaluation, and Reflection

The keycard access system works reliably from the areas that I've been able to test. It has worked having each part of the system far apart, as well as close up. It should be scalable in a room, but more testing will be needed to see how the addition of other systems, wireless networks, and human interference affect the messaging signals. Being that this is an IoT device and the code is open source, it will need to be more secure in the future.

Conclusions and Future Work

In order to make further progress on this project, I would work on a few key areas. One is the implementation of MAC Address whitelisting on the Pi wireless network, in order to limit connections in addition to hiding that network. I would also look to implement firewalls to limit port access on the NodeMCUs and the Pi. There is room for improvement on the scripting on the NodeMCUs as well, in order to make the code more efficient, and more scalable. Some suggestions would be to make more use of variables in place of hard code, in order to reproduce it on multiple readers without much end user interaction.

Bibliography

Programs Used:

Arduino IDE: https://www.arduino.cc/en/main/software

Python: https://www.python.org/downloads/

Mosquitto MQTT: https://mosquitto.org/

MySQL: https://www.mysql.com/downloads/

Protocols Used:

MQTT: http://mqtt.org/

ODBC: https://docs.microsoft.com/en-us/sql/odbc/microsoft-open-database-connectivity-odbc?view=sql-server-ver15

Guides and Technical Help:

Pi as a WAP: https://pimylifeup.com/raspberry-pi-wireless-access-point/

Remote MySQL: https://howtoraspberrypi.com/enable-mysql-remote-connection-raspberry-pi/

Using ODBC with Filemaker: https://fmhelp.filemaker.com/docs/edition/en/fm_odbc_jdbc_guide.pdf

MQTT with Pi and NodeMCU: https://www.instructables.com/id/How-to-Use-MQTT-With-the-Raspberry-Pi-and-ESP8266/

MF RC522 Arduino library: https://github.com/miguelbalboa/rfid/wiki/Useful-code-snippets#detection-of-tag-removal

Helpful Arduino Discussion: https://forum.arduino.cc/index.php?topic=177602.0

Appendices

Appendix 1: Pi Python script

```
import datetime
import mysql.connector
import paho.mqtt.client as mqtt

# Don't forget to change the variables for the MQTT broker!
mqtt_username = "administrator"
mqtt_password = "*****"
mqtt_topic = "pi1/+/keycardauth"
time_topic = "pi1/+/timeSync"
mqtt_broker_ip = "192.168.4.1"

client = mqtt.Client()
# Set the username and password for the MQTT client
client.username_pw_set(mqtt_username, mqtt_password)

# These functions handle what happens when the MQTT client connects
# to the broker, and what happens then the topic receives a message
def on_connect(client, userdata, flags, rc):
    # rc is the error code returned when connecting to the broker
    print ("Connected!", str(rc))

    # Once the client has connected to the broker, subscribe to the topic
    client.subscribe(mqtt_topic)
    client.subscribe(time_topic)

def send_on(device,accessid):
    powerTopic = "relay" + device + "/power"
    accessTopic = "relay" + device + "/access"

    client.publish(powerTopic,1)
    client.publish(accessTopic,accessid)

def send_off(device):
    powerTopic = "relay"+ device +"/power"
    client.publish(powerTopic,0)

def send_time(device):
    time = int(datetime.datetime.now().timestamp())
    #adjust for local timezone/daylight savings (Currently Daylight savings so GMT-4)
    time = time - 14400

    #If not Daylight Savings (GMT-5)
    #time = time - 18000

    #add for message lag
    time = time + 5
    timeTopic = "relay" + device + "/time"
    client.publish(timeTopic,time)

def on_message(client, userdata, msg):
    # This function is called everytime the topic is published to.
    # If you want to check each message, and do something depending on
    # the content, the code to do this should be run in this function
    db = mysql.connector.connect(
        host="localhost",
        user="connector",
        passwd="*****",
        database="pi1"
    )

    topic = msg.topic
    device = topic.split('/')[1].strip()
    deviceNumber = device[-1]

    if("timeSync" in topic):
        print ("Time sync received")
        send_time(deviceNumber)
    else:
        uid = msg.payload.decode('utf-8')
        cardID = str(uid)

        print (device + " " + cardID)

        cursor = db.cursor()

        cursor.execute("SELECT accessid FROM keycardaccess WHERE keycardid = %s AND
equipmentid = %s", (cardID, deviceNumber) )
        queryResult = str(cursor.fetchone())
        print (queryResult)

        if queryResult == "None":
           exists = 0
        else:
           accessid = int(queryResult[1])
           exists = 1

        if exists >= 1:
           print ("Equipment access found for this card")
           if accessid == 1:
              start = time(8,00)
              end = time(20,00)
           elif accessid == 2:
              start = datetime.time(7,00)
              end = datetime.time(22,00)
           elif accessid == 3:
              start = time(8,00)
              end = time(17,00)
           elif accessid == 4:
              start = time(0,00)
              end = time(24,59,59)

           current = datetime.datetime.now().time()

           print (str(current))

           if current >= start and current <= end:
              print ("Current time is during access time")
              send_time(deviceNumber)
              send_on(deviceNumber,accessid)
              onoff = 1
```

```
else:
            print ("Current time is outside of access time")
            send_off(deviceNumber)
            onoff = 0
      else:
         print ("No equipment allows access from this card")
         send_off(deviceNumber)
         onoff = 0

      # Creates the log in the database
      sql_insert = "INSERT INTO accesslog (keycardid,equipmentid,onoff) VALUES (%s, %s, %s)"
      cursor.execute(sql_insert, (cardID, device, onoff))
      db.commit()

# Here, we are telling the client which functions are to be run
# on connecting, and on receiving a message
client.on_connect = on_connect
client.on_message = on_message

# Once everything has been set up, we can (finally) connect to the broker
# 1883 is the listener port that the MQTT broker is using
client.connect(mqtt_broker_ip, 1883)

# Once we have told the client to connect, let the client object run itself
client.loop_forever()
client.disconnect()
```

Appendix 2: Arduino Reader Code

```
#include <MFRC522.h>
#include <PubSubClient.h>
#include <ESP8266WiFi.h>

const char* ssid     = "PiTest";
const char* wifi_password = "******";

const char* mqttServer = "192.168.4.1";
const char* mqttTopic = "pi1/reader1/keycardauth";
const char* mqttTopic2 = "pi1/reader1/cardremoved";
const char* accessTopic = "reader1/access";
const char* mqttUser = "administrator";
const char* mqttPassword = "*******";
const int mqttPort = 1883;
const char* clientID = "reader1";

#define RST_PIN         0
#define SS_PIN          2
#define buzzer          4
#define redlight        5
#define greenlight      15

MFRC522 mfrc522(SS_PIN, RST_PIN);
WiFiClient espClient;
```

```
PubSubClient client(espClient);

bool rfid_tag_present_prev = false;
bool rfid_tag_present = false;
int _rfid_error_counter = 0;
bool _tag_found = false;

String cardID;

void ReceivedMessage(char* topic, byte* payload, unsigned int length) {
  if ((char)payload[0] == '0') {
    digitalWrite(greenlight, LOW);
    digitalWrite(redlight, HIGH);
  }
}

bool Connect() {
  if (client.connect(clientID, mqttUser, mqttPassword)) {
    client.subscribe(accessTopic);
    return true;
  }
  else {
    return false;
  }
}

void setup() {
  SPI.begin();
  mfrc522.PCD_Init();

  pinMode(buzzer, OUTPUT);
  pinMode(redlight, OUTPUT);
  pinMode(greenlight,OUTPUT);
  digitalWrite(redlight, HIGH);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid,wifi_password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
  }

  client.setServer(mqttServer,mqttPort);

  while(!client.connected()) {
    if (client.connect(clientID, mqttUser, mqttPassword)){
    }
    else {
      delay(200);
    }
  }

  client.subscribe(accessTopic);
  client.setCallback(ReceivedMessage);
}
```

```
void getID(){
  rfid_tag_present_prev = rfid_tag_present;
  _rfid_error_counter += 1;
  if(_rfid_error_counter > 2){
    _tag_found = false;
  }

  byte bufferATQA[2];
  byte bufferSize = sizeof(bufferATQA);
  mfrc522.PCD_WriteRegister(mfrc522.TxModeReg, 0x00);
  mfrc522.PCD_WriteRegister(mfrc522.RxModeReg, 0x00);
  mfrc522.PCD_WriteRegister(mfrc522.ModWidthReg, 0x26);

  MFRC522::StatusCode result = mfrc522.PICC_RequestA(bufferATQA, &bufferSize);

  if(result == mfrc522.STATUS_OK){
    if ( ! mfrc522.PICC_ReadCardSerial()){
      return;
    }
    _rfid_error_counter = 0;
    _tag_found = true;
  }
  rfid_tag_present = _tag_found;

  if(rfid_tag_present && !rfid_tag_present_prev){
    tone(buzzer, 2000); // Send 1KHz sound signal...
    digitalWrite(greenlight, HIGH);
    digitalWrite(redlight, LOW);
    delay(400);        // ...for 1 sec
    noTone(buzzer);     // Stop sound...
    cardID = "";

    for (byte i = 0; i < mfrc522.uid.size; i++) {
      cardID += String(mfrc522.uid.uidByte[i] < 0x10 ? "0" : "");
      cardID += String(mfrc522.uid.uidByte[i], HEX);
    }

    int str_len = cardID.length()+1;
    char payload[str_len];
    cardID.toCharArray(payload, str_len);

    client.publish(mqttTopic, payload);
    cardID = "";
  }
  if(!rfid_tag_present && rfid_tag_present_prev){
    digitalWrite(greenlight,LOW);
    digitalWrite(redlight,HIGH);
    client.publish(mqttTopic2,"0");
  }
}

void loop() {
 if(WiFi.status() != WL_CONNECTED){
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid,wifi_password);
    while (WiFi.status() != WL_CONNECTED) { // Wait for the Wi-Fi to connect
      delay(1000);
    }
  }

  if(!client.connected()){
    Connect();
  }

  client.loop();

  getID();

}
```

Appendix 3: Arduino Relay Code

```
#include <Time.h>
#include <TimeLib.h>
#include <PubSubClient.h>       // Include PubSubClient for MQTT talk
#include <ESP8266WiFi.h>        // Include the Wi-Fi library

const char* ssid     = "PiTest";
const char* wifi_password = "*******";

const char* clientID = "relay1";
const char* mqttServer = "192.168.4.1";
const char* mqttTopic = "relay1/power";
const char* mqttTopic2 = "relay1/access";
const char* mqttTopic3 = "relay1/time";
const char* mqttUser = "administrator";
const char* mqttPassword = "*********";
const int mqttPort = 1883;

#define relay      5

WiFiClient espClient;
PubSubClient client(espClient);

int startTime = 000;
int endTime = 2459;
int currentTime = 0;
int onoff = 0;

void ReceivedMessage(char* topic, byte* payload, unsigned int length) {
  if(strcmp(topic,mqttTopic) == 0){
    if ((char)payload[0] == '1') {
      digitalWrite(relay, HIGH);
      onoff = 1;
    }
    if ((char)payload[0] == '0') {
      digitalWrite(relay, LOW);
      onoff = 0;
    }
  }

  if(strcmp(topic,mqttTopic2) == 0){
    if ((char)payload[0] == '1') {
      startTime = 80000;
      endTime = 200000;
    }
    if ((char)payload[0] == '2') {
      startTime = 70000;
      endTime = 220000;
    }
    if ((char)payload[0] == '3') {
      startTime = 80000;
      endTime = 170000;
    }
    if ((char)payload[0] == '4') {
      startTime = 00000;
      endTime = 245959;
    }
  }

  if(strcmp(topic,mqttTopic3) == 0){
    payload[length] = '\0';
    String s = String((char*)payload);
    int i = s.toInt();
    setTime(i);
    }
}

bool Connect() {
  if (client.connect(clientID, mqttUser, mqttPassword)) {
    client.subscribe(mqttTopic);
    client.subscribe(mqttTopic2);
    client.subscribe(mqttTopic3);
    return true;
  }
  else {
    return false;
  }
}

void setup() {
  pinMode(relay,OUTPUT);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid,wifi_password);

  int i = 0;
  while (WiFi.status() != WL_CONNECTED) { // Wait for the Wi-Fi to connect
    delay(1000);
  }

  client.setServer(mqttServer,mqttPort);

  while(!client.connected()) {
    if (client.connect(clientID, mqttUser, mqttPassword)){
    }
    else {
      delay(200);
    }
  }
  client.subscribe(mqttTopic);
  client.subscribe(mqttTopic2);
  client.subscribe(mqttTopic3);

  client.setCallback(ReceivedMessage);

  client.publish("pi1/relay1/timeSync","1");
}

void loop() {
if(WiFi.status() != WL_CONNECTED){
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid,wifi_password);
    int i = 0;
    while (WiFi.status() != WL_CONNECTED) { // Wait for the Wi-Fi to connect
      delay(1000);
    }
  }

  if (!client.connected()) {
    Connect();
  }

  client.loop();

  time_t t = now();
  currentTime = (hour(t)*10000 + minute(t)*100 + second(t));

  if((currentTime < startTime or currentTime > endTime) and onoff == 1){
    digitalWrite(relay, LOW);
    onoff = 0;
    }
}
```