Abertay University

Password-less 2-Factor Authentication

Password-less 2-Factor Authentication using RF and facial recognition Technology.

Adam Board

CMP408: IoT and Cloud Secure Development

BSc Ethical Hacking Year 4

2023/24

Note that Information contained in this document is for educational purposes.

Contents

1 INTRODUCTION

1.1 PURPOSE

Keeping an employee's account secure is important for an organisation since it reduces the chance of a data breach occurring. In 2023 alone, over 40% of data breaches happened due to the use of stolen or weak credentials (Verizon, 2023). Choosing passwords puts a heavy responsibility on the user to create a strong password and remember it. Over 33% of users require multiple login attempts due to forgetting their password (Viral Parmar, 2022). Even if the password is strong, it can be easily stolen by an attacker in a phishing attempt.

The purpose of this project was to implement a method of password-less authentication with the combined use of Radio Frequency Identification (RFID) and Facial recognition as the two-step method to ensure account security is unique and personal. Removing the password from the authentication process reduces the chance of a user's account being breached through brute-forcing or phishing attempts. The user is required to scan their RFID card against the scanner, then wait for the photo to be taken. Once the photo is taken, it is uploaded to AWS' Rekognition API, where it is compared against an existing photo of the individual to confirm the user. After confirming the user, a code is given to unlock the website and provide the user their personal information.

1.2 OBJECTIVES

To complete this project, several objectives have been outlined for completion:

- A Kernel module to register a button press.
- A Kernel module to allow interaction with the LED and Buzzer through the interface.
- A Python Script on Raspberry Pi which can do the following:
    - Scan RFID card to check for correct card before activating the camera.
    - Take a photo using python's camera module and upload photo to AWS Rekognition API to compare against existing photo of individual on AWS.
    - If the photos match, provide generated code to unlock the webpage.
- Create an EC2 Web Server using Flask which:
    - Allows Raspberry Pi user account to input the matching generated code to unlock information on webpage.
    - Allows the webpage to display personal details of the user from S3 bucket.

2 PROCEDURE

2.1 OVERVIEW OF PROCEDURE

The two core sections of the project are the:

- Raspberry Pi setup which entails:
  - Kernel Modules
  - Physical Hardware.
  - Python Script utilizing the Rekognition API.
- AWS setup entails:
  - Creating IAM users.
  - Creating an S3 bucket
  - Creating an EC2 instance which runs a Flask webserver.

2.2 RASPBERRY PI

An overview of the process that occurs on the Raspberry Pi can be seen in the Flow Diagram in Figure 1.

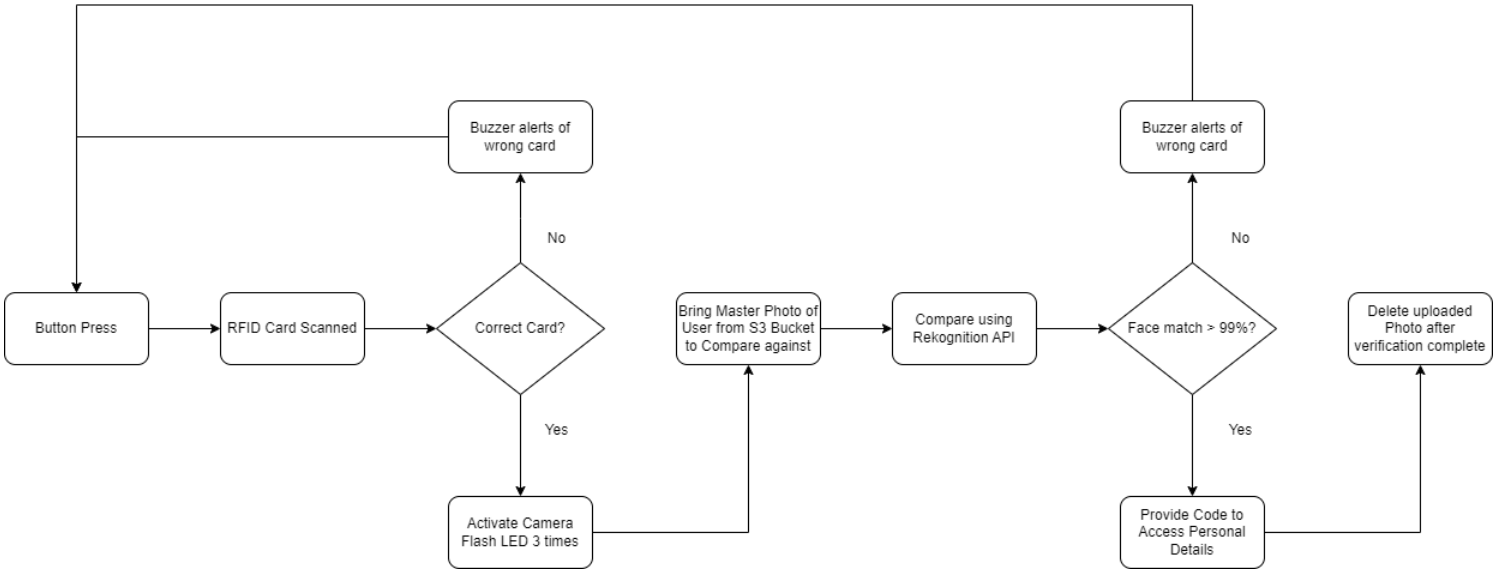

Figure 1 Flow Diagram of Raspberry Pi execution.

2.2.1 Secure Configuration for Pi

Before developing the project, the Raspberry Pi required initial configuration. First, SSH was configured to establish remote access to the Raspberry Pi. The default password on the Raspberry Pi was changed

to prevent unauthorized attackers from gaining access to the system due to weak credentials (PassWarden, 2023). The password change can be seen in Figure 2.

```
pi@raspberrypi:~ $ passwd
Changing password for pi.
Current password:
New password:
Retype new password:
passwd: password updated successfully
pi@raspberrypi:~ $
```

Figure 2 Changing the default password on Raspberry Pi.

2.2.2 Hardware

Using a breadboard, The Raspberry Pi Zero was connected to an RFID Scanner (SPI interface, RST, MISO, MOSI, SCK, SDA), a blue LED (GPIO 13), a Buzzer (GPIO 21), a button (GPIO 15), and a camera module, that was connected through the CSI connector on the side of the Raspberry Pi. All the connections for hardware can be seen in Figure 3

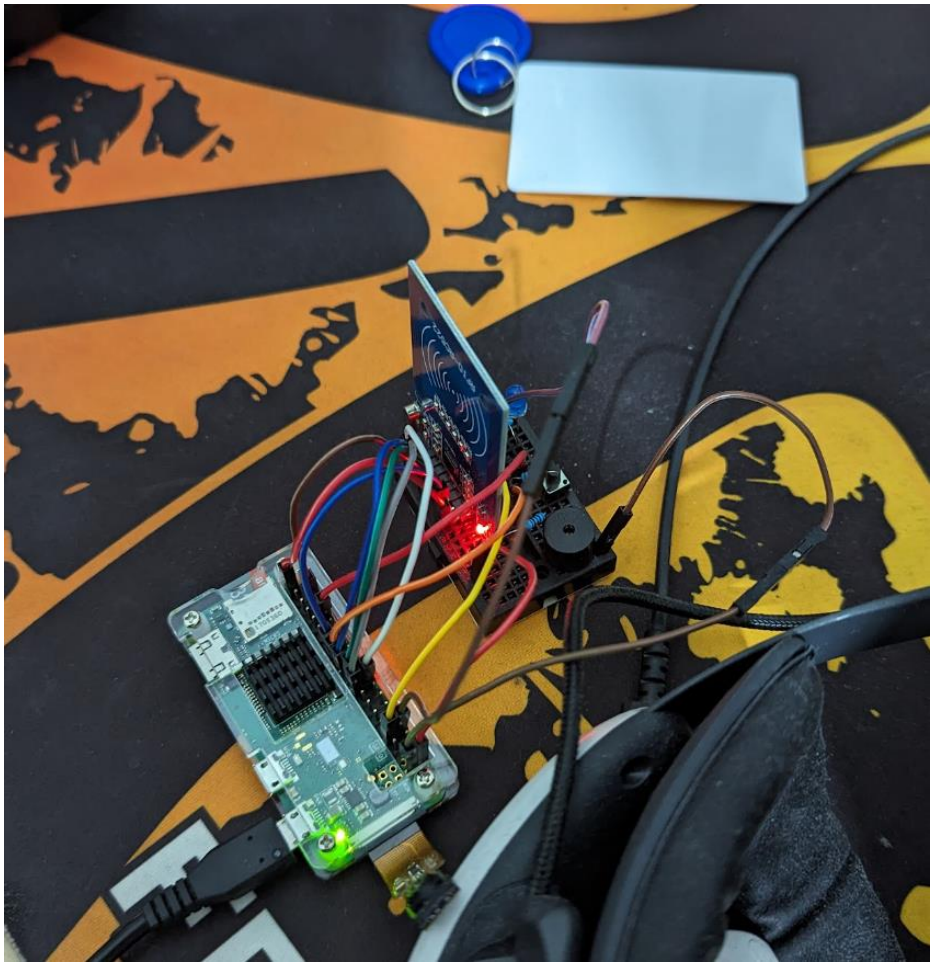

Figure 3 Raspberry Pi Hardware Setup.

2.2.3 Kernel Modules

To interface with the various hardware connected, two Kernel modules were implemented. Both Kernel modules were written in the programming language C and were cross compiled into Kernel modules suitable for the Raspberry Pi. They were transferred to the Raspberry Pi using the command "scp"

- piirq.ko: When the button connected at GPIO15 is pressed, the text "button_pressed" is printed into the Kernel log for the Python script to see. (piirq.c in Appendix A).
- piio.ko: Rather than using Python to interact directly with the pins, using this LKM allows the application "piiotest" to be called from the "Passwordless-Auth.py" Python script to interact with the Kernel module (piio.c in Appendix B).

2.2.4 Passwordless-Auth.py

The Python script "Passwordless-Auth.py" (Passwordless-Auth.py in Appendix C) checks the Kernel log for the text "button_pressed" to determine when to start the authentication process. Once pressed, the Python library, "MFRC522" will activate the RFID scanner, allowing a user to read their RFID card. If the information inside the card is correct, a blue LED will flash 3 times and the camera will activate to take the user's photo.

Once the photo is taken, it is compared against an original photo of the user using Rekognition's "compare_faces" function. This returns a confidence score to measure the similarity between the same face in two different pictures. To ensure accuracy in the facial recognition, a confidence score of 99% or above in similarity is required to pass the second step in the authentication process (Amazon, 2023).

The Python script interacts with Rekognition and other AWS services using the library "boto3", which is an SDK provided by Amazon for development with Python (Amazon, 2023).

If the user fails to pass any part of the authentication process, the buzzer will alert them by beeping three times and forcing the user to redo the current stage in the authentication process.

After passing both stages of the authentication process, the blue LED will glow for 5 seconds and the script will provide a code generated using the python library "uuid" (GeeksForGeeks, 2018). This code is then inserted into a web application which grants the user access to their personal information on the S3 bucket. Once the entire process is complete, the photo taken is removed from the device to reduce storage usage.

2.3 AMAZON WEB SERVICES

2.3.1 Users – IAM

Two user accounts were created, one for the Raspberry Pi that is running the Python script, and one for the Flask web application. It is always best practice to create accounts with the principle of Least-Privilege required to function. This ensures that if the account gets breached, the potential damage is mitigated (Microsoft, 2023).

The Raspberry Pi user account required permission to access Rekognition's "compare_faces" function, and the ability to add objects to the implemented S3 bucket.

The Flask web application user account required permission to list and get objects from the implemented S3 bucket. Both accounts policies can be seen in Figures 4 and 5.

piuser

The mininum required permissions for this account

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "rekognition:CompareFaces",
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": [
                "s3:PutObject",
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::adamstorage/*"
        }
    ]
}
```

Figure 5 Policies for Pi user account.

flaskwebappolicies

These are the mininum permissions required for the related user to function

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::adamstorage",
                "arn:aws:s3:::adamstorage/*"
            ]
        }
    ]
}
```

Figure 4 Policies for Flask web application user account.

2.3.2 S3 Bucket

The S3 bucket named "adamstorage" stored the comparison image and the personal information of the user. This bucket was made private to ensure only specific IAM users could access the objects. The files stored can be seen in Figure 6.

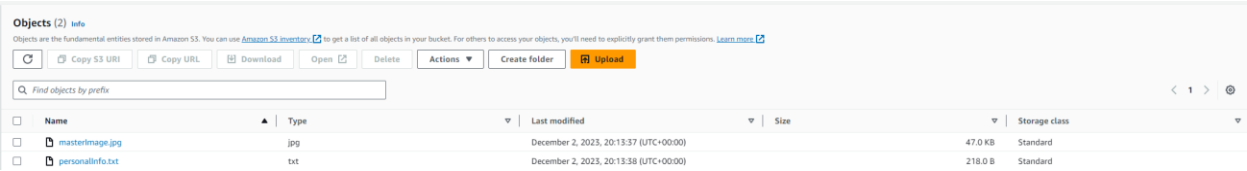

Figure 6 Stored objects inside of S3 bucket.

2.3.3 EC2 – Ubuntu Instance

A t2.micro EC2 instance running Ubuntu was set up. To allow access to the instance, SSH keys were created. At creation, the security group allowed SSH, HTTP, and HTTPS. However, the HTTP option was closed once the web application was working as intended and an SSL certificate was provided for HTTPS. This has been implemented to ensure traffic is encrypted during transport to mitigate On-path attacks (CloudFlare, 2021).

A Python Flask web application named "Server.py"(Appendix D) was set up on the Ubuntu Instance to allow a user access to their personal information (GO-FOR-IT, 2022). On the main webpage, "index.html" (Appendix E), a user could insert a code, which is given through the script on the Raspberry Pi. This can

be seen in Figure 7. There was input validation to check that the formatting of the input was the same as the generated code's format. Afterwards, the input is checked to find a matching code within the S3 bucket. If a match is made, three presigned_URLs are generated to allow the second webpage, "DataDisplay.html", to display the personal information of the user for a limit of 5 minutes (CloudFlare, 2023). To interact with the S3 Bucket, a role with the Flask user policies was attached to the EC2 instance. The output of a successful code being entered can be seen in Figure 8.

Figure 7 index.html with place to input generated code.

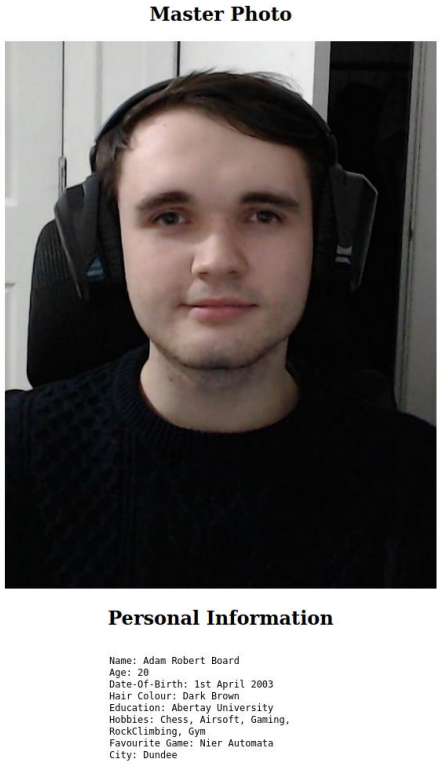

Figure 8 Output of successful authentication with generated code.

The SSL certificate was generated automatically using Caddy, which is shown in Figure 9, and Figure 10 shows the proof of the SSL certificate working.

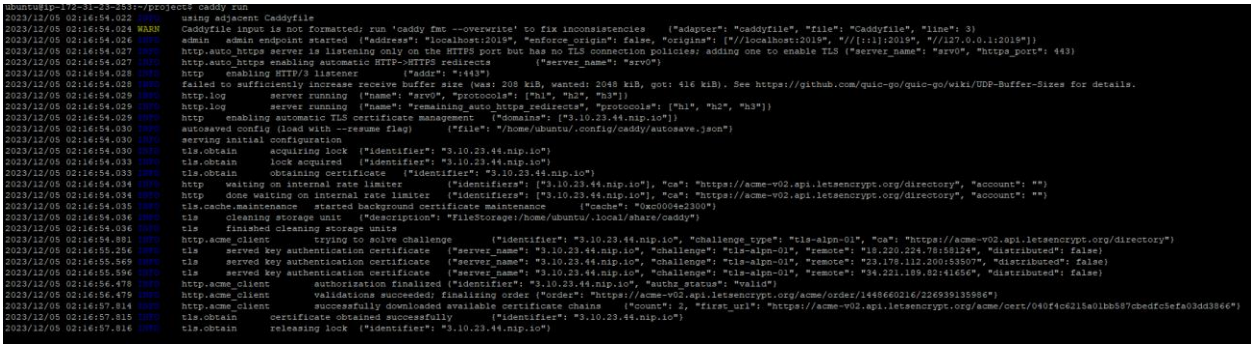

Figure 9 Code to automatically generate SSL certificate upon running reverse proxy.

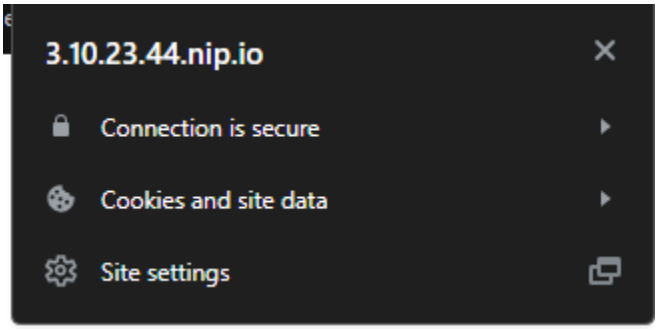

Figure 10 Proof of secure connection.

3 DISCUSSION

3.1 CONCLUSION

This project successfully implemented a password-less authentication process using RFID and facial recognition to allow a user to access their sensitive personal data. However, the current project only allows one user to use the authentication system in total. While this is a limitation, the proof of concept has still been achieved.

Various pieces of hardware are controlled via Python libraries and Kernel modules. The “mfrcc522” library allows the RFID scanner to be interacted with using Python code. Although, the library only accepts Mifare Classic RFID tags, which is an issue for compatibility. “Boto3” allows the Python code to interact with Amazon Web services - including the S3 bucket which stores the user’s information, the EC2 instance hosting the web application, and the Rekognition API for facial recognition.

During the development process, security was factored into each part of the project. The web application used an SSL certificate, the S3 bucket was kept private, and different IAM users were created with separate permissions for the Python script and the web application.

3.2 FUTURE WORK

The project could be improved by implementing a lifecycle rule into the S3 bucket to delete generated codes after a period of time. This would reduce the total storage being used by unnecessary files and would aid in lowering the cost of running the authentication project in the future.

Additionally, the web application could be improved by implementing the ability to logout, which would end the expiration timer on the presigned URLs early, allowing users more control over their personal data and how long it would be displayed. Finally, improving the design and aesthetic of the web application would improve the overall user experience when using the web application.

REFERENCES

Amazon, 2023. *Comparing Faces in Images.* [Online] Available at: https://docs.aws.amazon.com/rekognition/latest/dg/faces-comparefaces.html [Accessed 25 November 2023].

Amazon, 2023. *Overview of face detection and face comparison.* [Online] Available at: https://docs.aws.amazon.com/rekognition/latest/dg/face-feature-differences.html [Accessed 25 November 2023].

CloudFlare, 2021. *Why is HTTP not Secure.* [Online] Available at: https://www.cloudflare.com/en-gb/learning/ssl/why-is-http-not-secure/ [Accessed 2 December 2023].

CloudFlare, 2023. *Presigned URLs.* [Online] Available at: https://developers.cloudflare.com/r2/api/s3/presigned-urls/ [Accessed 2 December 2023].

GeeksForGeeks, 2018. *Generating random id's using UUID in Python.* [Online] Available at: https://www.geeksforgeeks.org/generating-random-ids-using-uuid-python/ [Accessed 25 November 2023].

GO-FOR-IT, 2022. *Deploying a Flask Application on EC2.* [Online] Available at: https://aws.plainenglish.io/deploying-a-flask-application-on-ec2-54cfeb396fa1 [Accessed 2 December 2023].

Microsoft, 2023. *Implementing Least-Privilege Administrative Models.* [Online] Available at: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/implementing-least-privilege-administrative-models [Accessed 1 December 2023].

PassWarden, 2023. *What is a Raspberry Pi Device.* [Online] Available at: https://www.passwarden.com/help/use-cases/raspberry-pi-default-password [Accessed 16 November 2023].

Verizon, 2023. *Results and Analysis: Introduction.* [Online] Available at: https://www.verizon.com/business/resources/reports/dbir/2023/results-and-analysis-intro/ [Accessed 15 November 2023].

Viral Parmar, H. A. S. R. H. P. A. S. P., 2022. *A Comprehensive Study on Passwordless Authentication.* s.l., Institute of Electrical and Electronics Engineers.

APPENDICES

APPENDIX A – PIIRQ.C

```
#include <linux/module.h>

#include <linux/gpio.h>
#include <linux/interrupt.h>

static unsigned int Button = 15;
static unsigned int Irqnum = 0;
static unsigned int Counter = 0;

static irq_handler_t piirq_irq_handler(unsigned int irq, void *dev_id, struct
pt_regs *regs){

   printk(KERN_INFO "button_press");
   Counter++;

   return (irq_handler_t) IRQ_HANDLED;
}

int __init piirq_init(void){
    int result = 0;
    pr_info("%s\n", __func__);
    //https://www.kernel.org/doc/Documentation/pinctrl.txt
    printk("piirq: IRQ Test");
    printk(KERN_INFO "piirq: Initializing driver\n");

    if (!gpio_is_valid(Button)){
    printk(KERN_INFO "piirq: invalid GPIO\n");
    return -ENODEV;}


       gpio_request(Button, "Button");
       gpio_direction_input(Button);
       gpio_set_debounce(Button, 200);
       gpio_export(Button, false);


    Irqnum = gpio_to_irq(Button);
    printk(KERN_INFO "piirq: The button is mapped to IRQ: %d\n", Irqnum);
```

```
    result = request_irq(Irqnum,
          (irq_handler_t) piirq_irq_handler, // pointer to the IRQ handler
          IRQF_TRIGGER_RISING,
          "piirq_handler",    // cat /proc/interrupts to identify
          NULL);

    printk("piirq loaded\n");
    return 0;
}
void __exit piirq_exit(void){

   free_irq(Irqnum, NULL);
   gpio_unexport(Button);
   gpio_free(Button);
   printk("piirq unloaded\n");
}
module_init(piirq_init);
module_exit(piirq_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Adam Board");
MODULE_DESCRIPTION("RPi Button Press detection");
MODULE_VERSION("0.5");
```

APPENDIX B – PIIO.C

```
#include "piio.h"
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/fs.h>
#include <asm/uaccess.h>
#include <linux/uaccess.h>
#include <linux/gpio.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/seq_file.h>

static int DevBusy = 0;
static int MajorNum = 100;
static struct class*  ClassName  = NULL;
static struct device* DeviceName = NULL;
```

```
lkm_data lkmdata;
gpio_pin apin;

static int device_open(struct inode *inode, struct file *file)
{
    printk(KERN_INFO "device_open(%p)\n", file);

    if (DevBusy)
        return -EBUSY;

    DevBusy++;
    try_module_get(THIS_MODULE);
    return 0;
}

static int device_release(struct inode *inode, struct file *file)
{
    printk(KERN_INFO " device_release(%p)\n", file);
    DevBusy--;

    module_put(THIS_MODULE);
    return 0;
}


static int      device_ioctl(struct file *file, unsigned int cmd, unsigned long
arg){
    int i;
    char *temp;
    char ch;

    printk("device_ioctl: Device IOCTL invoked : 0x%x - %u\n" , cmd , cmd);

    switch (cmd) {
    case IOCTL_PIIO_READ:
        strcpy(lkmdata.data ,"This is from lkm\0");
        lkmdata.len = 101;
        lkmdata.type = 'r';

        copy_to_user((void *)arg, &lkmdata, sizeof(lkm_data));
        printk("Device IOCTL WRITE\n");
        break;
```

```
    case IOCTL_PIIO_WRITE:
        printk("Device IOCTL READ\n");
        copy_from_user(&lkmdata, (lkm_data *)arg, sizeof(lkm_data));
        printk("From user: %s - %u - %c\n" , lkmdata.data , lkmdata.len ,
lkmdata.type);
        break;

    case IOCTL_PIIO_GPIO_READ:
        memset(&apin , 0, sizeof(apin));
        copy_from_user(&apin, (gpio_pin *)arg, sizeof(gpio_pin));
        gpio_request(apin.pin, apin.desc);
        apin.value = gpio_get_value(apin.pin);

        strcpy(apin.desc, "LKMpin");

        copy_to_user((void *)arg, &apin, sizeof(gpio_pin));
        printk("piio IOCTL_PIIO_GPIO_READ: pi:%u - val:%i - desc:%s\n" , apin.pin
, apin.value , apin.desc);
        break;
    case IOCTL_PIIO_GPIO_WRITE:

        copy_from_user(&apin, (gpio_pin *)arg, sizeof(gpio_pin));
        gpio_request(apin.pin, apin.desc);
        gpio_direction_output(apin.pin, 0);
        gpio_set_value(apin.pin, apin.value);

        printk("piio IOCTL_PIIO_GPIO_WRITE: pi:%u - val:%i - desc:%s\n" ,
apin.pin , apin.value , apin.desc);


        break;
    default:
            printk("piio: command format error\n");
    }

    return 0;
}


struct file_operations Fops = {
    .unlocked_ioctl = device_ioctl,
    .open = device_open,
    .release = device_release,
};
```

```
static int __init piio_init(void){
    int ret_val;
    ret_val = 0;

       printk(KERN_INFO "piio: Initializing the piio\n");
       MajorNum = register_chrdev(0, DEVICE_NAME, &Fops);
          if (MajorNum<0){
             printk(KERN_ALERT "piio failed to register a major number\n");
             return MajorNum;
          }
       printk(KERN_INFO "piio: registered with major number %d\n", MajorNum);


       ClassName = class_create(THIS_MODULE, CLASS_NAME);
       if (IS_ERR(ClassName)){
          unregister_chrdev(MajorNum, DEVICE_NAME);
          printk(KERN_ALERT "Failed to register device class\n");
          return PTR_ERR(ClassName);
       }
       printk(KERN_INFO "piio: device class registered\n");


       DeviceName = device_create(ClassName, NULL, MKDEV(MajorNum, 0), NULL,
DEVICE_NAME);
       if (IS_ERR(DeviceName)){
          class_destroy(ClassName);
          unregister_chrdev(MajorNum, DEVICE_NAME);
          printk(KERN_ALERT "Failed to create the device\n");
          return PTR_ERR(DeviceName);
       }
       printk(KERN_INFO "piio: device class created\n");


    return 0;

}

static void __exit piio_exit(void){
       device_destroy(ClassName, MKDEV(MajorNum, 0));
       class_unregister(ClassName);
       class_destroy(ClassName);
       unregister_chrdev(MajorNum, DEVICE_NAME);
       gpio_free(apin.pin);
```

```
	printk(KERN_INFO "piio: Module removed\n");
}
module_init(piio_init);
module_exit(piio_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Adam Board");
MODULE_DESCRIPTION("RPi GPIO Interact Driver");
MODULE_VERSION("0.2");
```

APPENDIX C – PASSWORDLESS-AUTH.PY

```
# ------------------------------------------------------------------------ #
#                    CMP408 IoT and Cloud Secure Development               #
# ------------------------------------------------------------------------ #
# ------------------------------------------------------------------------ #
#                             Adam Board - 2005335                         #
# ------------------------------------------------------------------------ #
# ------------------------------------------------------------------------ #
#             Mini Project - Passwordless 2-Factor Authentication          #
# ------------------------------------------------------------------------ #


import RPi.GPIO as GPIO
import time
from mfrc522 import SimpleMFRC522
from picamera import PiCamera
from PIL import Image
import os
import subprocess as sub
import csv
import boto3
import uuid
from urllib import parse

# -------------------- Initialisation of Global Variables -------------------- #
Phase1Validation = False
Phase2Validation = False
begin = False

# ----------------- Basic initialisation of Devices on System ---------------- #
text = 0
Buzzer = 21
```

```
Scanner = SimpleMFRC522()
CamLight = 13
Button = 15
Camera = PiCamera() #csi connector



# ------------------ Cleanup to ensure the text are the same ----------------- #

def CleanText(unclean):
    nospaces = unclean.replace(" ", "")
    clean = nospaces.lower()
    return clean

# ------------------------ Grab AWS Creds from file ------------------------ #

print("Accessing Login Details")
with open("Pi_accessKeys.csv", "r") as file:
    reader = csv.reader(file)
    for row in reader:
        accessKeyID = row[0]
        secretAccessKey = row[1]


# ------------------------ Grab RFID code from file ------------------------ #
print("Accessting RFID Code")
with open("Authentication.csv", "r") as file:
    reader = csv.reader(file)
    for row in reader:
        Auth = row[0]
        Auth = CleanText(Auth)
        break


# ---------------------- Setting up Rekognition client ---------------------- #
print("initialising access to Rekognition client")
clientRek = boto3.client("rekognition", region_name="eu-west-2",
aws_access_key_id=accessKeyID, aws_secret_access_key=secretAccessKey)
print("Rekognition connected")


# ------------------------ setting up the S3 client ------------------------ #
print("Initialising access to S3 Bucket client")
clientS3 = boto3.client('s3', aws_access_key_id=accessKeyID,
aws_secret_access_key=secretAccessKey)
```

```
bucket = "adamstorage"

print("S3 Connected!")



# --------- RFID Card authentication as initial authentication stage --------- #
def RFIDValidation():
    print("Beginning RF Validation, Please Present Card...")
    id, text = Scanner.read()
    print("Here is our details")
    print(text)
    text = CleanText(text)
    if text == Auth:
        authentication = True
        GPIO.cleanup()
        return authentication

    else:
        authentication = False
        GPIO.cleanup()
        return authentication



# --------- Validation using facial recognition from Rekognition API --------- #
def PhotoValidation():
    print("Beginning Photo, Please Point Camera Towards You!")
    time.sleep(3)
    print("Light will start flashing, after three flashes, camera will take
photo!")
    for i in range (3):
        os.system("./piiotest writepin 13 1")
        time.sleep(1)
        os.system("./piiotest writepin 13 0")
        time.sleep(1)
    Camera.capture('img_Valid.jpg')
    print("photo has been taken")
    photoTaken = Image.open('img_Valid.jpg')
    photoTaken.save('img_Valid.jpg')
    matchAuth = CompareFaces()
    if matchAuth == True:
```

```
        authentication = True
        print("True has been chosen!")
        return authentication

    else:
        authentication = False
        print("False has been chosen")
        return authentication

# ------------ Function for comparing faces using Rekognition API ------------ #
def CompareFaces():

    clientS3.download_file(bucket,"masterImage.jpg","masterImage.jpg")
    print("master image downloaded, beginning to open files")
    time.sleep(2)
    imageSource = open("masterImage.jpg", "rb")
    imageTarget = open("img_Valid.jpg", "rb")

    print("both files have been opened, beginning comparison")
    time.sleep(2)
    response = clientRek.compare_faces(SimilarityThreshold=99,
SourceImage={"Bytes": imageSource.read()}, TargetImage={"Bytes":
imageTarget.read()})

    for faceMatch in response["FaceMatches"]:
        similarity = str(faceMatch["Similarity"])
        print("face matches with " + similarity + "% confidence")
        time.sleep(4)
        if faceMatch["Similarity"] >= 99:
            print("Face Matches!")
            imageSource.close()
            imageTarget.close()
            os.remove("masterImage.jpg")
            return True

        else:
            print("Faces do not match!")
            imageSource.close()
            imageTarget.close()
            os.remove("masterImage.jpg")
            return False
```

```
# ------- Uploads the randomly generated code to allow the user access ------- #
def UnlockAWS():
    code = uuid.uuid4().hex.upper()[0:8]
    codeFile = open("staging.txt", "w")
    codeFile.write("Access Granted")
    codeFile.close()
    print(str(code))
    deleteTags = {"Delete": ""}
    clientS3.upload_file("staging.txt", bucket, str(code) + ".txt")

    print("Here is The Code to your Personal Information, Enjoy!")
    time.sleep(20)
    CleanUp()



# ------------------ Alerts user of incorrect authentication ----------------- #
def IncorrectID():
    print("Wrong ID/Photo, Please Try again!")
    os.system("./piiotest writepin 21 0")
    time.sleep(2)
    for i in range (3):
        os.system("./piiotest writepin 21 1")
        time.sleep(0.5)
        os.system("./piiotest writepin 21 0")
        time.sleep(0.5)
    Phase1Validation = False
    Phase2Validation = False


def CleanUp():

        os.system("./piiotest writepin 13 0")
        os.system("./piiotest writepin 21 0")
        print("System Cleanup Complete! Have a Nice Day!")
        GPIO.cleanup()
        exit()




# ------------------------------------------------------------------------- #
#                            ^Function code above^                          #
```

```
# ------------------------------------------------------------------------ #

# ------------------------------------------------------------------------ #
#                          Starting point of program                       #
# ------------------------------------------------------------------------ #

print("ready to begin Authentication process, please press the button!")


#will keep the user here until they press the button to continue to the
authentication
while begin == False:
    #Runs 'cat /proc/kmsg' and redirects output to a pipe
    p = sub.Popen(('cat', '/proc/kmsg'), stdout=sub.PIPE)



    for row in iter(p.stdout.readline, b''):
        row = row.decode()
        if "button_press" in row:
            print("button pressed! be ready to scan")
            begin = True
            break

while Phase1Validation == False and Phase2Validation == False:
# ---------------- initiates the validation process with RFID ---------------- #

#if incorrect then it will play incorrect noise and require a rescan of the card
    while Phase1Validation == False:
      Phase1Validation = RFIDValidation()
      print(Phase1Validation)
      if Phase1Validation == False:
          IncorrectID()

# ---------------- Initiates the validation process with face ---------------- #

#If the correct card is used, will begin second validation with photo
#Which will be uploaded to AWS for analysis with Rekognition
    print("Passed Phase 1, Commencing Phase 2...")
    while Phase2Validation == False:
        Phase2Validation = PhotoValidation()
        print(Phase2Validation)
        if Phase2Validation == False:
            IncorrectID()
```

```
os.system("./piiotest writepin 13 1")
time.sleep(5)
os.system("./piiotest writepin 13 0")
UnlockAWS()
CleanUp()
```

APPENDIX D – SERVER.PY

```
from flask import Flask, render_template, request, redirect, send_file
import boto3
from botocore.client import Config
import re
import csv

app = Flask(__name__)

# ------------------------- setting up the S3 client ------------------------- #
s3Client = boto3.client('s3',config=Config(signature_version="s3v4"))

bucket = "adamstorage"


# ----------------- Function to gain access to personal data ----------------- #

def AccessInfo(bucket, accessCode):

#Initialising the variable before using it
    generateURLAccess = 1
    public_urls = []

#Using presigned URLs gives secure access to the information within the S3 Bucket
which expires to ensure it cannot be kept open longer than 5 minutes

    try:
        generateURLAccess = s3Client.generate_presigned_url("get_object", Params
=
    {"Bucket": bucket, "Key": accessCode + ".txt" }, ExpiresIn = 300)

    except Exception as e:
        pass
```

```
    try:
        generateURLImage = s3Client.generate_presigned_url("get_object", Params =
    {"Bucket": bucket, "Key": "personalInfo.txt" }, ExpiresIn = 300)

    except Exception as e:
        pass


    try:
        generateURLInfo = s3Client.generate_presigned_url("get_object", Params =
    {"Bucket": bucket, "Key": "masterImage.jpg" }, ExpiresIn = 300)

    except Exception as e:
        pass

    print(generateURLAccess)
    print(generateURLImage)
    print(generateURLInfo)

    return generateURLAccess, generateURLImage, generateURLInfo

# ---------------------- Function and route for homepage --------------------- #

@app.route("/")
def index():
    return render_template("index.html")

# ------------------ Function and route for data processing ------------------ #

@app.route("/dataProcess", methods=["POST"])
def DataProcess():

    #Gathering the code from the input
    accessCode = request.form.get("insertCode")

    #Pattern for matching validation for code that has been inserted to ensure
other files are not grabbed
    validation = r'^[a-zA-Z0-9]{8}$'

    #Checks to ensure the access code meets the requirements and isn't a
injection attempt
    if re.match(validation, accessCode):
        matchResult = s3Client.list_objects_v2(Bucket=bucket, Prefix=accessCode)
```

```
        information = 0
        image = 0

        if 'Contents' in matchResult:
            #if the code exists
            contents, image, information = AccessInfo(bucket, accessCode)
        else:
            #tell user that their input does not exist
            text = "Input is not valid and this does not exist"
            return render_template('index.html', text=text)

        if contents == 0:
            text = "cannot find anything, please insert something"
            return render_template('index.html', text=text)
        else:
            #display the user's information on different page
            return render_template('DataDisplay.html', Image = information,
Information = image)
    else:
        #informs the user that the code entered is invalid
        text = "The code inserted is invalid"
        return render_template('index.html', text=text)

if __name__ == '__main__':
    app.run(debug=True)
```

APPENDIX E – INDEX.HTML

```
<!DOCTYPE html>
<html>
<head>
<title>Passwordless-Auth</title>
</head>
<body style="text-align: center;">

<h1>Please Complete Authentication Process to Access Personal Data!</h1>
<h2>Once you have the code, please insert into the form!</h2>
<form method="POST" action="/dataProcess">
    <input type="text" name="insertCode" placeholder="Enter Generated Code">
    <input type="submit" value="Submit">
    </form>
```

```
    <div>
        {% if text%}
        <h1 style="color: red;">{{ text }}</h1>
        {% endif %}

    </div>

</body>
```

APPENDIX F – DATADISPLAY.HTML

```
<!DOCTYPE html>
<html>
<head>
<title>Personal Information</title>
</head>
<body style="text-align: center;">

    <div>
        <h2>Master Photo</h2>

        {% if Image %}
        <img src={{ Image }} style="width: 550px;">
        {% endif %}
        <h2>Personal Information</h2>
        {% if Information %}
        <embed src= {{ Information }}>
        {% endif %}
    </div>


</body>
```