AIR-GAPPED FACIAL RECOGNITION

Anvay Buch

Dr. Juola

Duquesne University

Abstract

The purpose of this project was to create a secure entry system for easier and safe access to my home lab environment that alleviates the main issue of memorizing a complex and long password. The solution I devised to solve this issue was implementing an air-gapped Raspberry Pi that has a webcam for facial recognition, which can connect via network-free SSH to my home lab. Overall the project works very well, allowing me secure entry into my lab environment in an expeditious manner with an average entry time of 3-4 seconds.

Introduction/Background

Biometric security is a growing field in the realm of cybersecurity solutions for identifying individuals for a variety of purposes including entry into an electronic device, entry into countries, and even entry into government buildings. It is a solution that provides both high security and assurance. As the goal of this project is to create a more secure facial recognition system for my home lab environment, utilizing biometric security was a must. However, this would not provide all the security needed, as Linux/Ubuntu does offer third-party facial recognition software from external vendors. The issue remains that we the user, are not aware of what is done with our data and biometrics on their end and how secure it truly is. By using a Raspberry Pi with a USB Webcam, I know that all the data processing is done on my own device in an offline state so no network attacks can steal such data or get into my home lab. Air gapping a device means that it is isolated from all networks and does not allow for any external wireless connections and this is exactly what I have done to my Raspberry Pi.

Materials

For this project, there were a minimal amount of materials required. These include-

Raspberry Pi 4B(RPi) - A low-cost, card-sized mini-computer that runs on a Linux-based operating system known as Raspbian OS.

OpenCV - An open-source python library based around real-time computer vision. This is the main library that will be used for my facial recognition software and model training.

USB Webcam - Using an old webcam I had lying around my house allows for a no-cost solution for video input to the facial recognition program. It will be connected to the Raspberry Pi

USB C to A Cable - The power and data cable that will both link the Raspberry Pi to my Homelab to allow for network-free SSH and provide power to the RPi.

Linux(Ubuntu) Homelab - The actual system that will be connecting to the Raspberry Pi solution over passwordless SSH for entry.

Methods

The first step I took in creating this project was the creation of the facial recognition software and model training. On the Raspberry Pi(referred to as RPi) I used the UNIX shell to clone the OpenCV repository to a folder labeled facial_recognition. The first program I wrote/adapted was a python script named headshots.py that detected the USB Webcam, created a folder with my name, and then took headshot pictures for the model to train from. This was adapted from a script that was included in the OpenCV package.

headshots.py

```
import cv2
name = 'Anvay'
cam = cv2.VideoCapture(0)
cv2.namedWindow("press space to take a photo", cv2.WINDOW_NORMAL)
cv2.resizeWindow("press space to take a photo", 500, 300)
img_counter = 0
while True:
  ret, frame = cam.read()
  if not ret:
    print("failed to grab frame")
    break
  cv2.imshow("press space to take a photo", frame)
  k = cv2.waitKey(1)
  if k % 256 == 27:
    # ESC pressed
    print("Escape hit, closing...")
    break
  elif k % 256 == 32:
    # SPACE pressed
    img_name = "dataset/" + name + "/image_{}.jpg".format(img_counter)
    cv2.imwrite(img_name, frame)
    print("{} written!".format(img_name))
```

```
    img_counter += 1
cam.release()
cv2.destroyAllWindows()
```

The OpenCV package also included a script that automatically trained the model named train_model.py. This was run once the headshot program got enough pictures (around 10-15) and would allow for the model to start recognizing my face automatically.

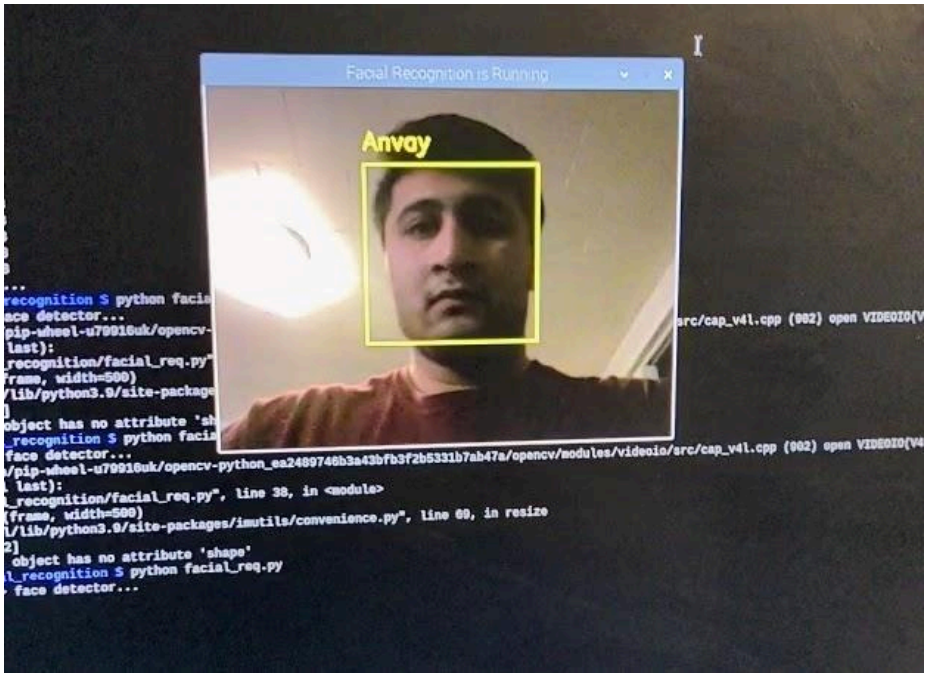
train_model.py

```
#! /usr/bin/python

# import the necessary packages
from imutils import paths
import face_recognition
#import argparse
import pickle
import cv2
import os

# our images are located in the dataset folder
print("[INFO] start processing faces...")
imagePaths = list(paths.list_images("dataset"))

# initialize the list of known encodings and known names
knownEncodings = []
knownNames = []

# loop over the image paths
for (i, imagePath) in enumerate(imagePaths):
  # extract the person name from the image path
  print("[INFO] processing image {}/{}".format(i + 1, len(imagePaths)))
  name = imagePath.split(os.path.sep)[-2]

  # load the input image and convert it from RGB (OpenCV ordering)
  # to dlib ordering (RGB)
  image = cv2.imread(imagePath)
  rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
  # detect the (x, y)-coordinates of the bounding boxes
  # corresponding to each face in the input image
  boxes = face_recognition.face_locations(rgb, model="hog")

  # compute the facial embedding for the face
  encodings = face_recognition.face_encodings(rgb, boxes)

  # loop over the encodings
  for encoding in encodings:
    # add each encoding + name to our set of known names and
    # encodings
    knownEncodings.append(encoding)
    knownNames.append(name)

# dump the facial encodings + names to disk
print("[INFO] serializing encodings...")
data = {"encodings": knownEncodings, "names": knownNames}
f = open("encodings.pickle", "wb")
f.write(pickle.dumps(data))
f.close()
```

The final python script that needed to be written/adapted was the actual facial recognition script. While OpenCV does come with its own variation of such a script, it did need to be changed to suit the purposes of this project. The OpenCV script iterated forever until the “q” key was entered to kill the program, however, I changed the script so that it ran 3-4 times and then quit if no name was recognized for both the purposes of speed and that the program needs to end at some point so login can continue(whether successful or not). The program includes a way to test if the training worked and if my face was detected then a square would be placed over my head with my name(*Figure 1*).

Figure 1



facial_req.py

```
from imutils.video import VideoStream
import face_recognition
import imutils
import pickle
import time
import cv2

#Initialize 'currentname' to trigger only when a new person is identified.
currentname = "unknown"

encodingsP = "encodings.pickle"

# load the known faces and embeddings

print("[INFO] loading encodings + face detector...")
data = pickle.loads(open(encodingsP, "rb").read())

# src = 0 : I had to set it to 0 in order to use the USB webcam attached to
```

```
my laptop
vs = VideoStream(src=0, framerate=10).start()

time.sleep(2.0)

# loop over frames from the video file stream
for x in range(0, 4):

  frame = vs.read()
  frame = imutils.resize(frame, width=500)
  # Detect the face boxes
  boxes = face_recognition.face_locations(frame)
  encodings = face_recognition.face_encodings(frame, boxes)
  names = []

  # loop over the facial embeddings
  for encoding in encodings:
    # attempt to match each face in the input image to our known
    matches = face_recognition.compare_faces(data["encodings"], encoding)
    name = "Unknown"

    # check to see if we have found a match
    if True in matches:
      matchedIdxs = [i for (i, b) in enumerate(matches) if b]
      counts = {}

      # loop over the matched indexes and maintain a count for
      # each recognized face face
      for i in matchedIdxs:
        name = data["names"][i]
        counts[name] = counts.get(name, 0) + 1

      name = max(counts, key=counts.get)

      #If someone in dataset is identified, print their name on the screen
      if currentname != name:
        currentname = name
        print(currentname)

    # update the list of names
    names.append(name)

  # loop over the recognized faces
```

```
  for ((top, right, bottom, left), name) in zip(boxes, names):
    # draw the predicted face name on the image - color is in BGR
    cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 225), 2)
    y = top - 15 if top - 15 > 15 else top + 15
    cv2.putText(frame, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX, .8,
                (0, 255, 255), 2)

  # display the image to our screen
  #cv2.imshow("Facial Recognition is Running", frame)
  key = cv2.waitKey(1) & 0xFF

# cleanup for memory
cv2.destroyAllWindows()
vs.stop()
```

The final step was to make sure that when I clicked on my user profile on my home lab environment that the password-based system was configured to default to the facial recognition platform. This was done by editing the /etc/profile folder to SSH into the RPi when my UID was selected. The SSH is done with no network, using the USB C to A cable that connected both the devices as it allows the RPi to be put into an OTG mode, essentially being detected as an IO device. The SSH was done using passwordless SSH and encrypted using RSA public/private key exchange. The keys were generated and distributed to both devices accordingly and allows for a secure connection that doesn't rely on the entering user to enter a password. The RPi was configured so that if an incoming SSH connection comes in that the facial recognition program is automatically run. The added code in /etc/profile checks to see if the output of the facial recognition code is my name using grep, if the name is found then the SSH connection ends, and the log-in continues as normal, otherwise, an error 1 command is sent and the home lab returns to the main login screen. The code also logs both successful and unsuccessful attempts with a timestamp in a log file on the system.

/etc/profile (*Only the additional code*)

```
if [ "`id -u -n`" == "anvay" ]; then
     ssh -tt tacwin@rpi4.local 2> /dev/null | grep -i "anvay" >
/dev/null
     if [ "$?" -ne "0" ]; then
          now=`date +"%D   %T"`
          echo "login failed at $now" >> /var/attempt_log.log
          exit 1
     else
          now=`date +"%D   %T"`
          echo "login successful at $now" >> /var/attempt_log.log
     fi
fi
```

Results

Overall the project as a whole was a great success as the facial recognition and login systems both work extremely well and quite quickly. The timing for log-in averages around 3-4 seconds I measured over 15 log-in attempts.

I will be using this solution as a permanent fixture for my home lab for the time being and hope to improve it further in the future.

Discussion

While the project does work extremely well as a solution to my initial problem, there are future upgrades and changes I would like to make. First I would like to improve the hardware of the actual device while still maintaining its small form factor. This would come by way of a memory upgrade for the RPi and a more modern and better webcam. This is in hopes of increasing the speed for an even more seamless login process. I would also like to implement a script that would take a picture and place that in the log file as well for any successful or unsuccessful attempt just for the purpose of a better reference of logins.

Conclusion

With the primary goal of this project to create a secure solution for biometric facial log-in to my home lab, it is clear to see that all those marks were reached. The RPi allows for an air-gapped external hardware device to do all the data processing offline for facial recognition and extra layers of defense for a system that needs to stay protected.

Works Cited

Gillis, Alexander S. “What Is an Air Gap?: Definition from TechTarget.” *WhatIs.com*, TechTarget, 13 Sept. 2022, www.techtarget.com/whatis/definition/air-gapping.

“Home.” *OpenCV*, 7 Dec. 2022, opencv.org/.

Libretexts. “02-E.16.1: System Wide User Profiles: /Etc/Profile.” *Engineering LibreTexts*, Libretexts, 13 July 2022, eng.libretexts.org/Bookshelves/Computer_Science/Operating_Systems/Linux_-_The_Penguin_Marches_On_(McClanahan)/02%3A_User_Group_Administration/5.03%3A_System_Wide_User_Profiles/5.03.1%3A_System_Wide_User_Profiles%3A_etc-profile#:~:text=The%20%2Fetc%2Fprofile%20contains%20Linux,settings%20are%20defined%20for%20users.

Linuxize. “How to Setup Passwordless SSH Login.” *Linuxize*, Linuxize, 19 Feb. 2019, linuxize.com/post/how-to-setup-passwordless-ssh-login/.

Opencv. “Opencv/Opencv: Open Source Computer Vision Library.” *GitHub*, github.com/opencv/opencv.

Raspberry Pi Foundation. “Teach, Learn, and Make with the Raspberry Pi Foundation.” *Raspberry Pi Foundation*, www.raspberrypi.org/.