Boston University
Electrical and Computer Engineering
EC464 Senior Design Project

2nd Prototype Test Report

Team 24
Community Security

Samuel Evans evanss@bu.edu
Esteban Hernandez hesteban@bu.edu
Daniel-John Morel djmorel@bu.edu
Steve Numata numatas8@bu.edu
Cameron MacDonald Surman csms@bu.edu

February 29th, 2020

1.0 Introduction

For our second prototype testing, work revolved entirely around our data stream and the connections that needed to be made between the camera, AWS, and the user website. These connections are fundamental to the overall system for they allow us to report on events, conduct analysis, and then provide that information to those within the community.

We broke our testing up into the different points the data would pass through. Our first test is focused on recording motion-based events, syncing with S3, and then storing those clips on AWS. While seemingly simplistic, there are multiple moving parts that needed to be addressed to make this data transfer organized and efficient. The second test addressed the next connection, AWS to the user website. This involves footage uploaded into S3 to trigger an AWS Lambda handling function to insert it into a database hosted by AWS RDS, and ensuring it can be extracted by the user interface within the web application. Finally, we tested the eventual implementation of license plate recognition software through test scripts to ensure that the tools we plan on using will function as we need them to.

2.0 Raspberry Pi & AWS S3 Data Connection

2.1 Test Setup

Connecting the Raspberry pi's videos to AWS S3 storage requires multiple components to work together. The three main components are the motionEye program to collect videos, the AWS S3 library to sync videos, and the software utility cron to schedule the sync command every 2 minutes. All of these components were installed on the pi, and configured to work with each other. This entire testing setup used the pi's cellular shield to illustrate that the transfer process could be achieved via cellular data.

With the software already installed on the pi, the remaining part of the test setup revolved around recording a video and tracking its transfer path. As such, we had the camera record our audience for about 10 seconds. We could observe that the pi was indeed recording video since it added a new video entry to its camera directory (video path /var/lib/motioneye/Camera1/02-25-2020). After giving the pi a couple minutes for cron to sync the camera directory, we then opened the AWS S3 console to show the listed video, and played the video in there to ensure that it was properly added to our cloud storage.

2.2 Data

Since this test looked at uploading videos to S3 storage, the measurable criteria is more binary by nature. By this we mean that a video recorded by the Raspberry pi can be sent to S3

storage and then viewed from there. Our live test with our audience was successfully transferred from the pi to S3 storage as we could play the video in the AWS S3 console via its linked object URL.

2.3 Conclusions

This test demonstrated that the pi can record motion-detected videos and routinely send them to S3 storage automatically. For our final product, the cron sync time will be adjusted to an hourly interval rather than the 2 minute interval. Increasing the interval time will help reduce power and data consumption on the pi.

Now that the motion-detected video transfer is solidified for the pi, the next step is to ensure a livestream connection from the pi to the website. AWS Kinesis was considered for the task, but the Kinesis process on the pi competed with the motionEye process for control over the camera module. Since the pi's operating system only allows one one process access to the camera module, for it is a single I/O resource, motionEye has to be disabled in order for Kinesis to perform the livestream. Now motionEye provides access to the camera's livestream via HTTP. However, the cellular shield only allows the pi to initialize communication to an external device (not on the same network) and not the other way around. This one-way communication initialization is done for cellular device security. In order to provide our website access to the livestream, we are looking into options like port forwarding or having the pi establish the HTTP communication with our website first, so that the system can bypass the cellular communication restriction.

3.0.1 Pulling Video From AWS to Display to User

3.1 Test Setup

Testing the setup for this two part test required that an AWS Lambda function be created and set to trigger on an S3 mp4 insertion, as well as a Postgresql database and an instance of the database on AWS RDS. Both the AWS Lambda function as well as django needs functionality that allows it to communicate to the database. The two part setup for this design test is intended to first confirm that the Lambda function will insert the object url from a video file uploaded to S3 into the Postgresql database in AWS RDS. The second is to ensure that it is properly extracted from the Django web application.

3.2 Data

Testing the lambda function using a scripted test using a selected mp4 object url confirmed that the lambda function inserted the url into the database. This was confirmed by

viewing the Django admin page which shows all database items, as well as by checking the actual database using pgAdmin4. With the S3 trigger setup, the same results were found, verifying that the Lambda function worked as intended. In addition, the web application was confirmed to properly pull video from the database by selecting a combination of date and time on the web application. The correct video was displayed in an i frame and this was cross referenced by viewing the footage in S3. In addition, the web application properly displays no result of a certain selected date has no available footage, or if a camera has no available footage.

3.3 Conclusions

The two tests both confirmed that uploading a file to S3 in mp4 format will trigger the Lambda function to insert it into the database, and that video footage in the database can be extracted by the Django web app by selecting the time and date of the footage instance. This successful test demonstrated that there is a full functional bridge between the raspberry pi camera and the web application, as the camera can be left to record motion video and it will automatically be uploaded to the web application.

3.0.2 Automatically Sending Data from S3 to Postgresql Database

3.1 Test Setup

The measurable criteria for this test is confirmation that the RDS Database has been uploaded properly according to data sent to it from AWS Lambda. In addition, assuming the functionality of the database to web application communication is valid, and a proper S3 trigger is set up, when the Raspberry pi camera module uploads a video to S3, Lambda should automatically update the database with an entry containing the object file's URL.

3.2 Data

The testing procedure is split into an isolated test case and an overall system test case. The isolated test case consists of running an AWS Lambda test case containing the URL to an existing object file in S3, running it, and confirming it has been added to the "polls_footage_time" table in the Postgresql database. The second test is to have the Raspberry pi camera module upload a video file, and use pgAdmin4 to confirm that the video URL has been entered into "polls_footage_time" table in the database. Consequently, if Test 3.0.1 passes, it should be viewable on the web application as well.

3.3 Conclusions

The test will demonstrate that when a video footage is uploaded to S3 in the form of an mp4 video, AWS lambda will automatically add the object file's URL to the Postgresql database hosted by RDS. The test does not yet account for automatically sorting the footage into the correct date, as it will have to be done manually.

4.0 Testing OpenALPR Scripts on Test Clips

4.1 Test Setup

The test setup will be to have a command that processes license plate images or video feed. For this test we use a mp4 file from footage of cars passing by at around 10mph. Then we run the alpr program that calls the OpenAPR API which uploads the mp4 file to the cloud, we can then verify that the license plates were processed by checking the OpenALPR cloud.

4.2 Data

The testing procedure will take a mp4 file and upload them to OpenALPR through a Python program. The Python program will call on the OpenALPR API that will then upload the mp4 file to the OpenALPR cloud to process the license plates. Once the images are processed, we can check the valuable information on OpenALPR through a JSON data file. The JSON data file includes the car's plate number, model, type, and color.

4.3 Conclusions

This test shows that the ALPR program that we run can accurately detect license plates and can then read and store the license plate number in a JSON data file. This test also demonstrated that the ALPR program can process mp4 files that our Raspberry Pi uploads in a reasonable runtime.