Abertay University

CMP408

System Internals and Cyber Security

Part 4 – Mini Project

Name: Angelo Josey Caldeira

Student Number: 1605007

Building a basic IoT home security system using a Raspberry Pi and Amazon Web Services

Introduction:

Home crime and burglary is an ever-growing worry in less developed or poverty-stricken areas of the world. Home security systems are not cheap nor accessible to most people. This project aims to build a basic IoT security system using a Raspberry Pi and Amazon Web Services (AWS). In the realm of system internals and cyber security, it is important that processing sensitive data is secure from a cloud, software and hardware level. The proposed system will enable visual access into a user home; therefore, must be equipped with best security practices to detour those with malicious intent from accessing the system.

As mentioned previously, AWS will be used to achieve several tasks which would normally require additional hardware. The low running cost and extreme scalability of several AWS services make it an ideal option for this project.

In order to achieve the aforementioned aim, the following objectives will be attempted:

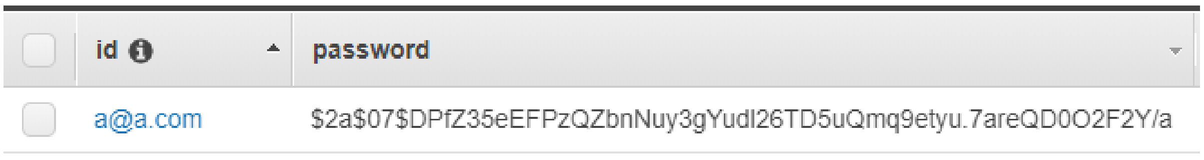
- Create a web interface with a secure login system
- Read a hashed password from AWS DynamoDB to authorise login
- Use the Raspberry Pi camera to take still images and display them to the user on demand.
- Store images securely in an AWS s3 bucket with a timestamp.
- Restrict access to AWS s3 based on IP address
- Utilise system interrupts to implement a 'panic button' feature which turns on the alarm.
- Subscribe to a secure MQTT broker to receive requests for initiating the alarm
- Publish to the MQTT broker to turn on the alarm
- Utilise system interrupts to silence the initiated alarm

Methodology:

Create web interface and login system:

The first step taken to develop the system was to install apache2 on the pi and create the web interface. The front-end consists of a simple layout, utilising Bootstrap for a responsive experience on all size devices. The popular MVC architecture was adopted, keeping database access, controllers and user views separate from each other (MDN Web Docs, 2019).

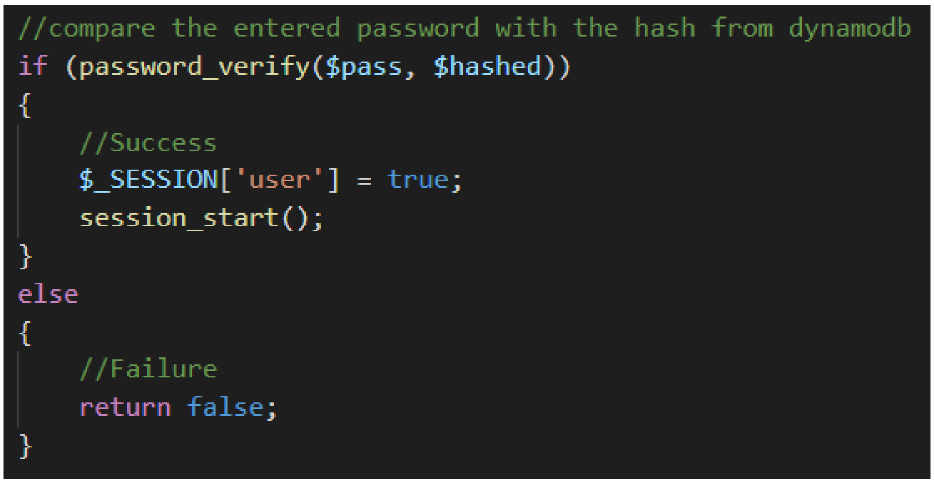
The web server is hosted locally on the Raspberry Pi and uses apache2 with php 7.3.11. The login system was designed using a server-side language – php. The possibility of SQL injection is mitigated as users are not allowed to write into the database. However, the application is required to read from the database and compare the stored password to the user's input. Figure 1 shows a default user that was manually input into the AWS DynamoDB alongside a hashed password. Php compares the stored hash against the user's input by using the *password_verify()* function.



id	password
a@a.com	$2a$07$DPfZ35eEFPzQZbnNuy3gYudl26TD5uQmq9etyu.7areQD0O2F2Y/a

Figure 1 - ID and hashed password in AWS DynamoDB

In order to manage user login and logout actions, the application leverages sessions. Sessions are an ideal choice when handling a login system as the sensitive information is stored on the server and cannot be accessed or altered by the user. The session will expire once the web page is closed and the user will be prompted with the login form again. A session will be started using session_start() if the *password_verify()* function returns true. This will allow the authorised user to access the web application. The code for this process can be seen in figure 2.



```php
//compare the entered password with the hash from dynamodb
if (password_verify($pass, $hashed))
{
    //Success
    $_SESSION['user'] = true;
    session_start();
}
else
{
    //Failure
    return false;
}
```

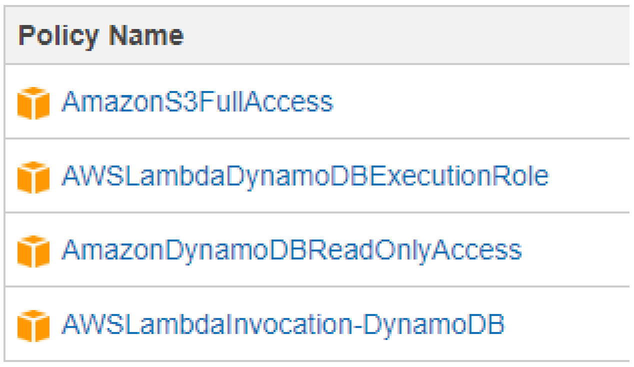
Figure 2 - Setting a session in php once the password has been validated

Additionally, a check is done to ensure the session is set and valid before running any functions through the web interface. This mitigates the risk of an attacker browsing directly to a php page on the server in order to execute a command. Furthermore, a blank index.php page has been added to each folder, ensuring the directory listing is hidden.

The web server includes a self-signed TLS certificate, ensuring all data passes through the application is encrypted (Digitalocean.com, 2019).

Interfacing with AWS:

Interfacing with AWS is done using awscli and the boto3 SDK for python (Amazon Web Services, Inc., 2019). In order to set up boto3, an IAM group must be created in AWS. Policies will be applied to the group, granting boto3 access to specific services. The group policies used for this project can be seen in Figure 3.



Policy Name
AmazonS3FullAccess
AWSLambdaDynamoDBExecutionRole
AmazonDynamoDBReadOnlyAccess
AWSLambdaInvocation-DynamoDB

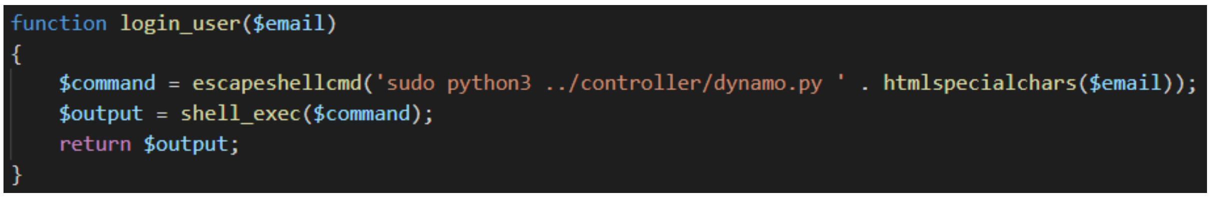
Figure 3 - IAM Group Policies

Once the group has been set up, the AWS Access key, AWS secret access key and region name must be provided during boto3 installation. This will allow the IAM user group to successfully interface with AWS through python. The application utilises boto3 for reading from DynamoDB and Uploading to an S3 Bucket.

Pi Camera and S3 Bucket

The Raspberry Pi camera is used to take still images on demand and display the results to the user. In order to achieve this, a python script must be executed on the Pi through. The python script is requested through php and executed using the *shell_exec()* function.

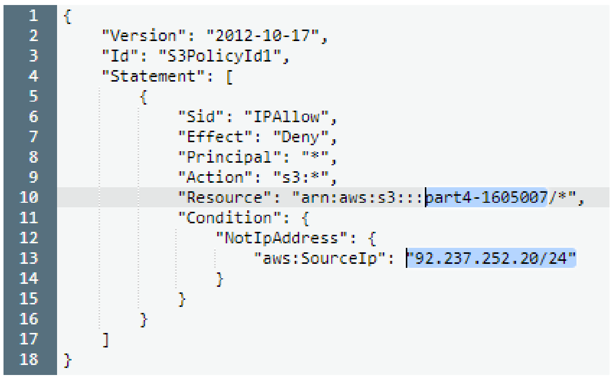
In order to ensure no malicious scripts are executed using this method, the intended command is put through *escapeshellcmd().* This function escapes characters that may be used to execute arbitrary commands (Php.net, 2019). Furthermore, the *htmlspecialchars()* function (with similar functionality) is run on all inputs that are appended to commands as seen in Figure 4.



```
function login_user($email)
{
    $command = escapeshellcmd('sudo python3 ../controller/dynamo.py ' . htmlspecialchars($email));
    $output = shell_exec($command);
    return $output;
}
```

Figure 4 - escapeshellcmd() and htmlspecialchars() functions being used on commands and user inputs

Images taken by the camera are instantly uploaded to AWS S3 using the boto3 SDK. To secure the S3 Bucket, a 'Bucket policy' was added (Docs.aws.amazon.com, 2019). This allows a specific IP or range of IP addresses to access public content in the bucket. The policy can be seen in Figure 5.



```
{
    "Version": "2012-10-17",
    "Id": "S3PolicyId1",
    "Statement": [
        {
            "Sid": "IPAllow",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:*",
            "Resource": "arn:aws:s3:::part4-1605007/*",
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": "92.237.252.20/24"
                }
            }
        }
    ]
}
```

Figure 5 - Bucket Policy to allow a single IP address

Interrupts:

The system includes a 'panic button' which is implemented by leveraging interrupts. Python is once again used for this feature – utilising RPi.GPIO. This library includes a function to detect a change in an electrical signal; *wait_for_edge()* – successfully blocking execution of the program without polling (Sourceforge.net, 2019). The python script for the 'panic button' is executed using *exec()* (alternative to *shell_exec()* where no result is returned). As mentioned previously all commands are escaped to ensure no malicious commands can be run.

Message Queuing Telemetry Transport (MQTT) Subscribe and Publish:

An MQTT broker is run on a micro EC2 instance which allows subscribing to and publishing messages. The server requires publishers and subscribers to authenticate using a username and password. Additionally, the broker includes TLS which keeps all messages encrypted and not readable to attackers. To further secure the EC2 instance, a security group was set up which specifies an IP address or range of IP addresses of which can access and communicate with the broker.

The system allows the user to subscribe to a single topic by selecting a checkbox and sending an AJAX request to php. The php page will execute a shell command with the specified topic. Future iterations of the system will allow subscribing to additional topics as only a single topic is currently available. Once the user has successfully subscribed to a topic, they are able to publish a message which, in turn, will activate the included buzzer (as an alarm). Furthermore, an interrupt is used at this point to silence the alarm by pressing the provided button. Figure 6 shows the web GUI of the aforementioned functionality. Figure 7 shows the AWS security group rules.

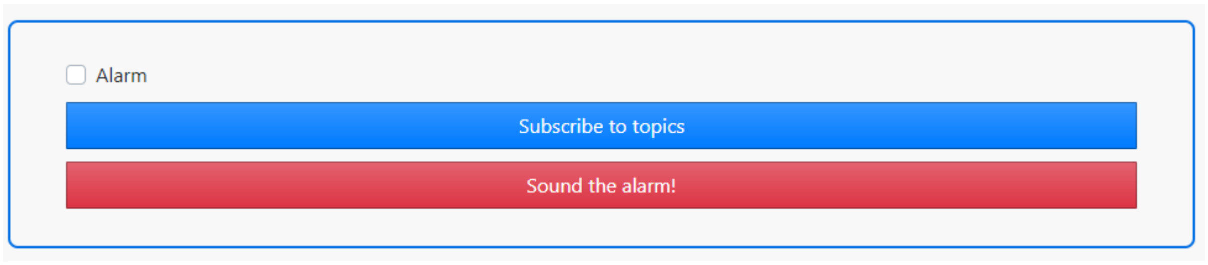
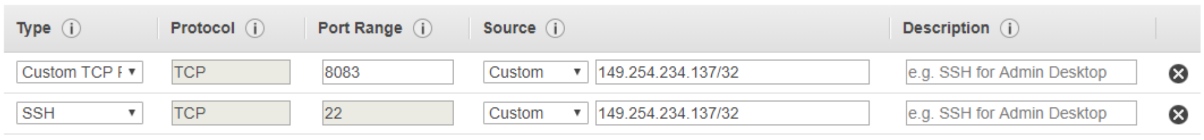


Figure 6 - Web GUI of MQTT subscribe and Publish functionality



Type	Protocol	Port Range	Source		Description	
Custom TCP F ▾	TCP	8083	Custom ▾	149.254.234.137/32	e.g. SSH for Admin Desktop	⊗
SSH ▾	TCP	22	Custom ▾	149.254.234.137/32	e.g. SSH for Admin Desktop	⊗

Figure 7 - EC2 instance security group

The project encompasses various aspects of system internals and cyber security; it aimed to build a basic IoT home security system on the Raspberry Pi while utilising cloud technologies through AWS. The system is composed of a camera, a buzzer (alarm) and a button which is used for interrupts – bringing high (web UI and cloud) and low level (interrupts) technologies together. Figure 8 shows a diagram the system interactions.

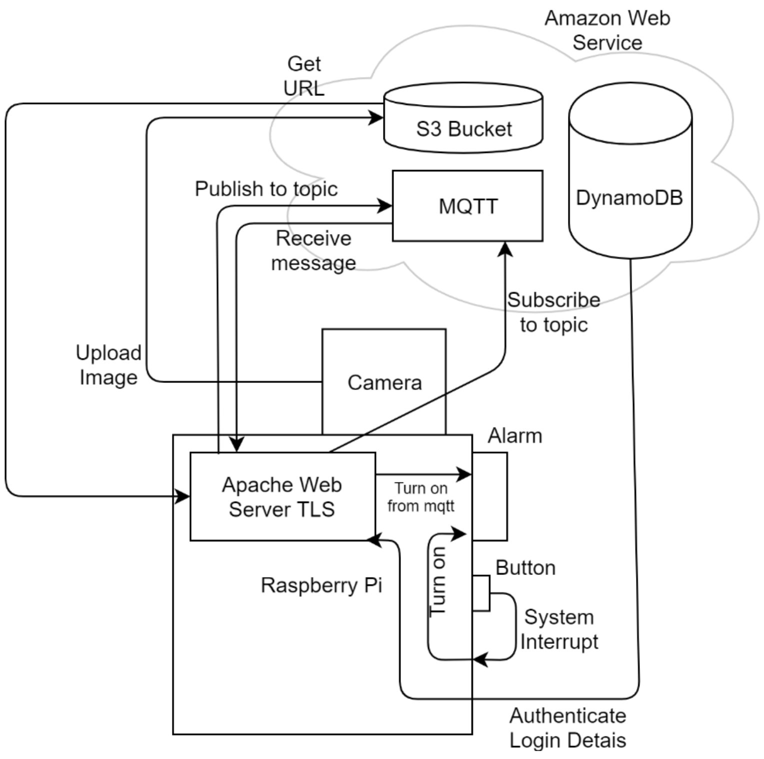


Figure 8 - Diagram of all system interactions

Conclusion:

The project successfully allows reading from DynamoDB for user authentication, uploading images to an S3 bucket and leveraging an EC2 instance as an MQTT broker. These functions are able to be executed by the end user through a web interface. The security aspect of the application was an extremely important consideration. Each layer of technology was inspected individually, applying best security practices where possible.

The cloud layer is secured through IP restrictions (security groups and bucket policy) and IAM user groups with restricted permissions. This will only allow specific users to have access and make changes to sensitive data. Software and hardware were secured using best practices such as secure sessions, string escaping, password hashing and an authentication process. Additionally, well known libraries such as RPI.GPIO were used (with best practice) to mitigate risk of GPIO pins being handled in an unexpected way.

References:

Amazon Web Services, Inc. (2019). *AWS SDK for Python*. [online] Available at: https://aws.amazon.com/sdk-for-python/ .

Digitalocean.com. (2019). *How To Create a Self-Signed SSL Certificate for Apache in Ubuntu 16.04 | DigitalOcean*. [online] Available at: https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-apache-in-ubuntu-16-04 .

Docs.aws.amazon.com. (2019). [online] Available at: https://docs.aws.amazon.com/AmazonS3/latest/dev//example-bucket-policies.html .

MDN Web Docs. (2019). *MVC*. [online] Available at: https://developer.mozilla.org/en-US/docs/Glossary/MVC .

Php.net. (2019). *PHP: escapeshellcmd - Manual*. [online] Available at: https://www.php.net/manual/en/function.escapeshellcmd.php .

Sourceforge.net. (2019). *raspberry-gpio-python / Wiki / Inputs*. [online] Available at: https://sourceforge.net/p/raspberry-gpio-python/wiki/Inputs/ .