

## Table of Contents

<b>Introduction .....</b>	<b>1</b>
<b>Relevance .....</b>	<b>1</b>
<b>Objectives.....</b>	<b>1</b>
<b>Methodology.....</b>	<b>2</b>
<b>Hardware .....</b>	<b>2</b>
<b>Software.....</b>	<b>3</b>
RFID Reader .....	3
Driver .....	3
Web Application .....	4
<b>Cloud.....</b>	<b>5</b>
Dynamo DB .....	5
Elastic Beanstalk .....	5
VPC.....	5
Security .....	6
<b>Conclusion .....</b>	<b>6</b>
<b>Bibliography.....</b>	<b>7</b>

## Introduction

### Relevance

MFA is already an effective security measure. Research has shown that having MFA enabled on an account makes it 99.9% less likely to be compromised, (Weinert, 2019). Adding an extra layer of authentication helps to keep these users secure. Most users use a mobile application that generates a code that is used to authenticate; however, a user can still be tricked into sending this code to a malicious actor, or simply be annoying users with the notifications, a strategy adopted by a Russian hacking group, (Cimpanu, 2021). Therefore, hardware tokens have a place within the security landscape for the most secure environments, or for locations where mobile devices are not allowed, like oil rigs. The project allows a user to use two different forms of authentication to access their personal account on a website. The first is the traditional “something-you-know” password, and the second is the “something-you-have” physical token. This token contains a unique identified (UID) that is used in a similar way as a traditional password.

### Objectives

The aim of the TAP-MFA project was to create a hardware-based authentication system along with a proof-of-concept website, to ultimately increase a websites security. The objectives of TAP-MFA were as follows:

1. Design and create software to read and process a RFID token.
2. Design and create a website that implements password and hardware token MFA.

3. Design and deploy cloud infrastructure to host the website and supporting database, making the system easily expandable.

## Methodology

### Hardware

The hardware used in this project was as follows:

- A Raspberry Pi Zero W<sup>1</sup>
- An SB Components RFID Reading HAT<sup>2</sup>
- 2x RFID tokens
- A breadboard
- One red and one green LED
- 2x 330-ohm resistors
- 4x male to female jumper wires
- A 16gb micro-SD card
- USB to micro-USB cable

The LEDs and resistors were connected to the breadboard and then to the GPIO passthrough pins on the RFID HAT. The fully assembled device can be seen in Figure 1.

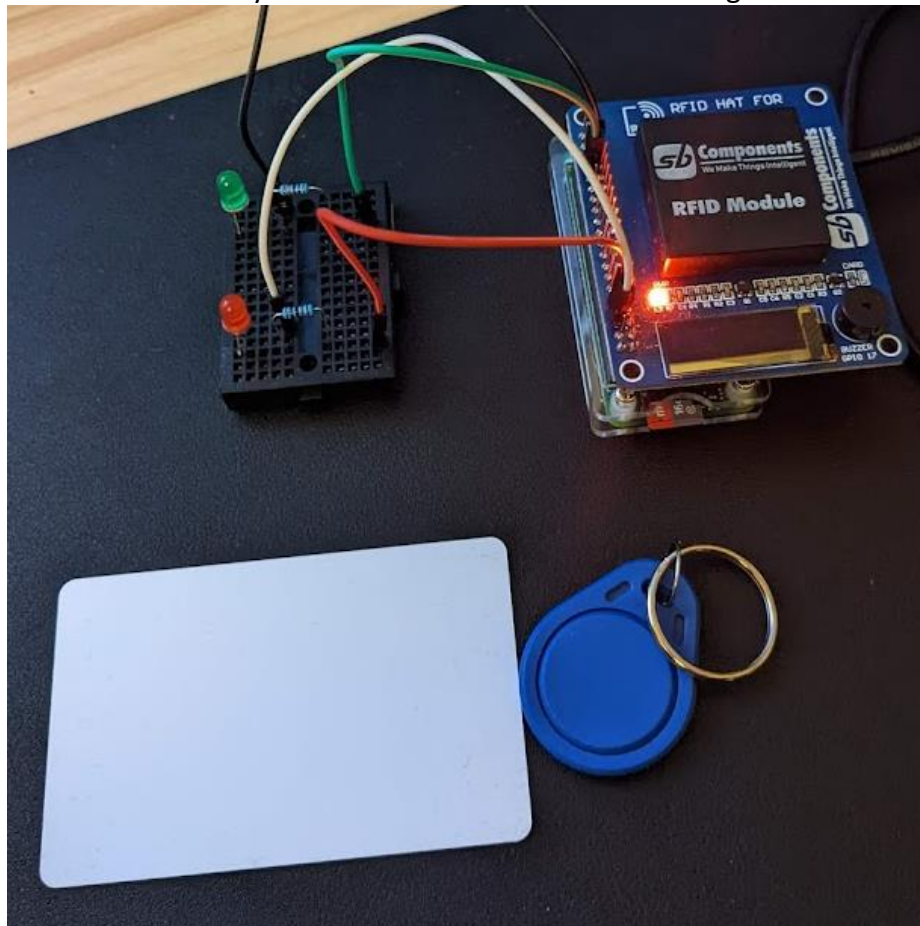


Figure 1: The fully assembled reader device

<sup>1</sup> <https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/>

<sup>2</sup> <https://thepihut.com/products/rfid-hat-for-raspberry-pi>

The RFID reading hat contained a small OLED screen and a buzzer, however, this buzzer was non-functional on arrival.

The green LED was connected to GPIO pin 12, and the red LED to GPIO pin 14. Both were also connected to a ground (GND) pin on the pi.

Each RFID token contained a UID as follows:

- Blue: 0796AF2EBDAD
- White: 0E0096CB94C7

## Software

### RFID Reader

The RFID HAT came with a python library that could read the UID from the token, and display text on the OLED screen, both of which were used in development. The sample application<sup>3</sup> provided by the developers was used as a starting point, however, this code was heavily modified.

The software ran a Flask webserver on port 5000 which responded to one GET request named 'getuid'. The webserver used HTTPS using a self-signed certificate. If put into production, a more appropriate service like Gunicorn would alongside an SSL certificate signed by an established certificate authority.

When an HTTPS GET request was sent to 'getuid', a function was run, and the green LED was enabled to indicate the device was ready to be used. Then, a token could be placed on the reader. Once placed, the UID was read from the token and the red LED was flashed to indicate the token had been read. The UID was then returned in a JSON format to the original requester.

During each stage of execution, messages were displayed on the OLED screen to communicate with the user.

### Driver

To control the LEDs via the GPIO pins, a Linux Kernel Module (LKM) was used. The LKM was a heavily modified version of an existing LKM. In its simplest form, the driver had a function that could be passed a struct. The structure indicated the GPIO pin number and the state to set the pin to, 1 for on and 0 for off. This code responsible for this can be seen in Figure 2. Using the 'fcntl' python library, the python script was able to make 'ioctl' calls to the driver to turn on or off different LEDs. This code snippet can be seen in Figure 3.

```
case IOCTL_PIIO_LED_CONTROL:
    copy_from_user(&apin, (gpio_pin *)arg, sizeof(gpio_pin));
    gpio_request(apin.pin, "control LED");
    gpio_direction_output(apin.pin, 0);
    gpio_set_value(apin.pin, apin.value);
    printk("pio: IOCTL_PIIO_LED_CONTROL: pi:%u - val:%i \n" , apin.pin , apin.value);
    break;
```

Figure 2: The lines of code responsible for changing the GPIO values within the LKM.

---

<sup>3</sup> <https://github.com/sbcshop/SB-RFID-HAT>

```
fd = open("/dev/piodev","wb") #driver

IOCTL_PIIO_LED_CONTROL = 105

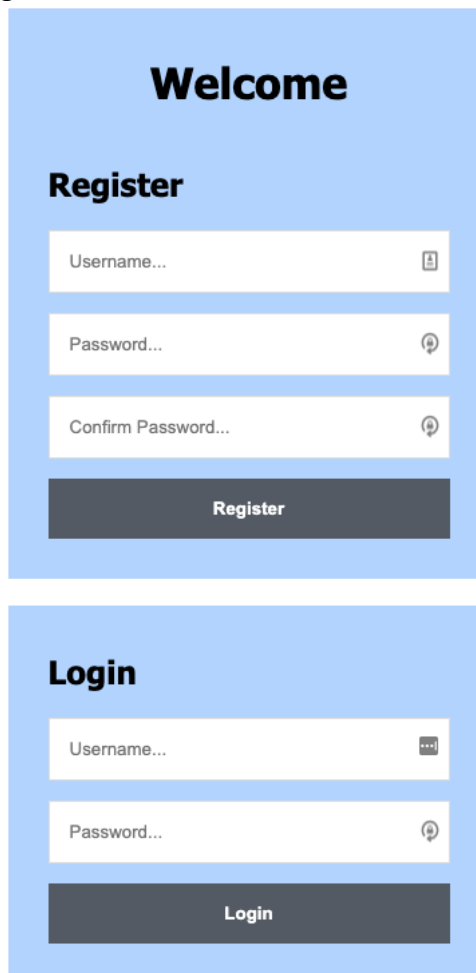
def switch_led(pin,state):
    pack = struct.pack('Ii',pin,state) #unsigned int, int
    ioctl(fd,IOCTL_PIIO_LED_CONTROL,pack)
```

Figure 3: A snippet of python code used to communicate with the driver.

## Web Application

The website was a Node application using the Express framework to handle HTTP requests. The initial design of the site was based on an example application created by David Adams, (Adams, 2020). His application was heavily modified, only following a similar HTML layout and a small amount of the Node code.

The landing page of the site consisted of two forms: a register form and a login form. A capture of the site can be seen in Figure 4.



The image shows two screenshots of a web application interface. The top screenshot is titled "Welcome" and features a "Register" form. The form has three input fields: "Username...", "Password...", and "Confirm Password...", each with a small icon to its right. Below the fields is a dark grey button labeled "Register". The bottom screenshot is titled "Login" and features a "Login" form with two input fields: "Username..." and "Password...", each with a small icon to its right. Below the fields is a dark grey button labeled "Login". Both forms are set against a light blue background.

Figure 4: The default register and login page

The application served four routes:

- / : The login and register page
- /login: Ran the login function

- /register: Ran the register function
- /home: A page the user was taken to after a successful sign in

The 'login' and 'register' routes had corresponding functions to handle their requests. The 'login' function took the username and password supplied by the user and checked it against the Dynamo Database using the AWS SDK, to authenticate the user. It first checked for the username inside the DB. If found, it would pull the username along with the BCrypt hashed UID and password and compared the retrieved password with a hashed version of the password supplied. If they matched, a request would be sent to the hardware device, and the retrieved UID would also be hashed and compared. If they matched, a session would be created for the user. If at any point there was a failure, the user was redirected back to the landing page.

The 'register' function works in a similar way but first checks the two entered passwords match before checking the DB for the username. Then, it obtains a UID from the hardware device, hashes the password and UID and stores them along with the username in the DB.

## Cloud

### Dynamo DB

A Dynamo DB was used to store user data for this project. Dynamo DB is a database service offered by AWS that uses NOSQL. This was a quicker and cheaper alternative to a traditional MySQL database which Amazon offers through RDS. This kept costs lower at the trade-off of some of the simplicity and structure offered by MySQL.

Dynamo DB held a single table, named 'db-tap-mfa', that stored the username, the UID of the MFA token, and the password. The UID and password were stored in their hashed form, as mentioned above.

### Elastic Beanstalk

Elastic Beanstalk is an AWS service that allows the quick configuration and deployment of web applications. In this project it was used to host the web application described earlier. The web application could then communicate with Dynamo DB using the AWS SDK to allow the application to function.

### VPC

A Virtual Private Cloud (VPC) was created to host the Elastic Beanstalk application. The thinking behind this was entirely future proofing and scalability. While the project could have functioned without the use of a VPC, creating one now means the more copies of the application could be hosted within it, and a load balancer could be used to handle heavy traffic loads.

The web application was hosted within the public subnet of the VPC, with a private subnet reserved for services not intended to be accessed by the internet. The web application was then accessed via an Elastic Network Interface which had a public IP address and handled the traffic between the internet and the private cloud. The cloud architecture has been visualised in the diagram below.

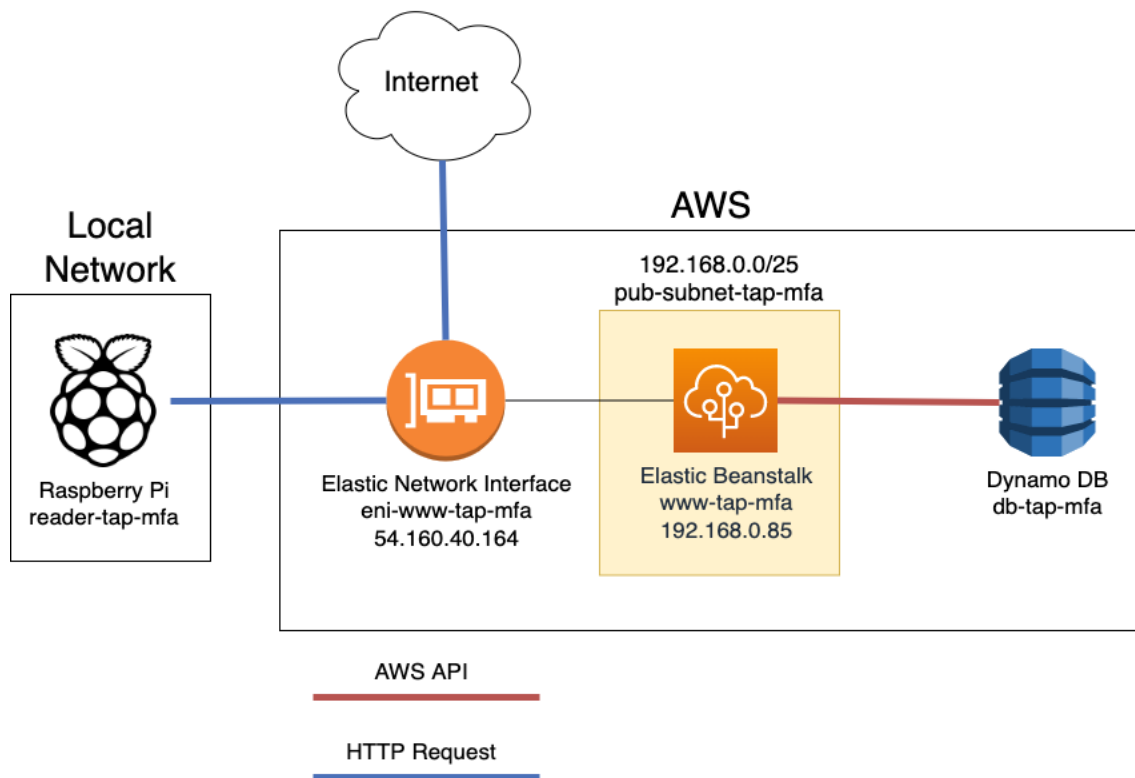


Figure 5: Network diagram of the cloud-based network and services, as well as how the hardware devices communicates with the services.

## Security

To keep the application secure, security groups were used to only allow the necessary inbound and outbound traffic to travel in and out the network. The SDK was also used to communicate securely between the web app and DynamoDB. IAM would have been used to create a specific user to handle communication between the web app and the DB, who only has permission to do such.

## Conclusion

This project has successfully demonstrated how a raspberry pi can be used as part of an authentication system for a cloud-based web application. While not a ground-breaking discovery, the low cost of equipment combined with the scalability of the cloud makes this project a great starting point for an expanded system that could be used by an enterprise to authenticate to internal web applications.

This project has met its aim fully, as demonstrated within this report, as well as the proof-of-concept video provided separately. The software and web application were both designed and developed successfully, and the whole project was deployed securely onto cloud infrastructure successfully.

## Bibliography

Adams, D., 2020. *Basic Login System with Node.js, Express, and MySQL*. [Online]

Available at: <https://codeshack.io/basic-login-system-nodejs-express-mysql/>

[Accessed 8 Jan 2022].

Cimpanu, C., 2021. *Russian hackers bypass 2FA by annoying victims with repeated push notifications*. [Online]

Available at: <https://therecord.media/russian-hackers-bypass-2fa-by-annoying-victims-with-repeated-push-notifications/>

[Accessed 7 Jan 2022].

Weinert, A., 2019. *Your Pa\$\$word doesn't matter*. [Online]

Available at: <https://techcommunity.microsoft.com/t5/azure-active-directory-identity/your-pa-word-doesn-t-matter/ba-p/731984>

[Accessed 7 Jan 2022].