CARDIFF UNIVERSITY

PRIFYSGOL CAERDYDD

Final Report
Project 140: "Internet of Things Security"

By Christopher Hutchings 1416357

CM3203 One Semester Individual Project, 40 Credits
Supervisor: Omer F Rana
Moderator: Kirill Sidorov

Cardiff University 2017

Contents

Table of Figures

Acknowledgements

I would like to personally thank Omer Rana, the supervisor of this project, for his help and support throughout the duration of this piece. I would also like to thank my family and friends for supporting me and keeping me motivated throughout.

1 Introduction

The Internet of Things (IoT) has made home automation, office automation, social media interaction and everyday tasks in life easier and simpler to carry out with the use of technology and IoT devices. Wi-Fi enabled kettles, central heating systems, baby monitors and cameras are just a small number of examples from the plethora of devices that can now be used and controlled through the Internet of Things via a smartphone device or other form of internet enabled device whilst at home or on the move.

Automation applications are becoming more and more popular as the IoT gains momentum making these devices even easier to control and link together through the use of conditional statements and logic. One of these applications is If-This-Then-That.

If-This-Then-That (IFTTT) is a popular automation application that allows a user to link two platforms or services together. For example, if tagged in a photo on Facebook, then that photo is uploaded to a folder on your Google Drive. It is a useful tool for managing and controlling social media, devices linked to your home network and cloud storage. However, when applications have access to lots of personal data as well as login credentials for third party applications, security can become an issue. This project focuses on the concept of making If-This-Then-That more secure through the introduction of a security layer between applets that are created using the application. Potentially dangerous people with malicious intentions can use automation applications to trigger attacks within a user's home or office through these Wi-Fi enabled devices.

The rationale and motivation for using IFTTT to assist in the autonomy of applications with IoT devices, as IFTTT provides the largest amount of services and platforms available on the current market. Despite the benefits of IFTTT, there is a potential vulnerability within their security which I aim to remedy with the implementation of the proposed solution below.

This project investigates these potential attacks and looks at implementing a layer that works at preventing them through the use of a policy engine that ensures an applet on IFTTT passes certain policy criterion before being fully executed to its usual extent. This report goes into detail about IFTTT itself, other current solutions available and the problem that is presented to users. This report details the approach taken to solve the problem and the implementation of this solution. The report also goes through the results of the solution and future work that could be carried out.

With the increasing prevalence of IoT enabled devices comes the additional requirement for enhanced security measures. By creating a verification process against a strict set of policy guidelines, this project will look to bridge the gap between increasingly sophisticated devices and natural persons' virtual and physical security.

1.1 Aim of the Project

The aim of this project is to create a web server that allows a user to add policies and rules to a file which will then be used as an added layer of security onto If-This-Then-That's applets. The solution that I will be implementing and discussing is a policy engine on how to deal with threats and security issues within the IoT, specifically using the If-This-Then-That application (IFTTT). The premise of the policy statement would be to clearly define accepted criteria for effective filtration of potentially harmful or malicious activities emanating from the exploitation of IoT devices.

1.2 Objectives

The general objective of this project is to create a way in which applets on the IFTTT application interact with a set of rules and policies set up by the user in order to add a more secure layer in-between the application and the user's device (mobile phone, house lights etc.)

The main objectives are to:

- Create and observe applets on the IFTTT application
  - Using Twitter and email services on the IFTTT application to demonstrate how security concerns can be overcome

- Learn how to setup a web server on a Raspberry Pi
  - Learn how to use the Flask web server framework on a Raspberry Pi so that the General Purpose Input/output pins can be remotely controlled via the web server

- Explore security policies that could be implemented
  - Look at policies such as verification of origin, only allowing certain time periods for applets to be used and encrypted channels

- Demonstrate how interaction with IFTTT, a Raspberry Pi and a web server can be used to simulate what could be done with other Internet of Things enabled devices
  - The solution should be easily transferrable to other devices such as Phillips Hue Lights in order to overcome security concerns for these devices

1.3 Policy Aims

The project intends to use created and customised policies that the solution will consult in order to filter out malicious or potentially harmful activities. The policies will be written into a configuration file that will be imported onto the server, restricting and preventing access or change from the outside world. Only the user can access and make changes to these policies.
An example of a scenario in which the policy can filter out unwanted commands and activities is shown in figure 1.1. If you have an oven which is controlled via IFTTT, you can put in place a policy that might say only turn the oven on after 5pm. Then if a command is sent to turn the oven on at 3pm, the server would consult the policy in place and would see it is 3pm, therefore do not turn on the oven and disregard this command. But if the time was 6pm and the command turn the oven on was sent to the server, it would consult the policy and say, yes it is after 5pm, therefore turn the oven on.

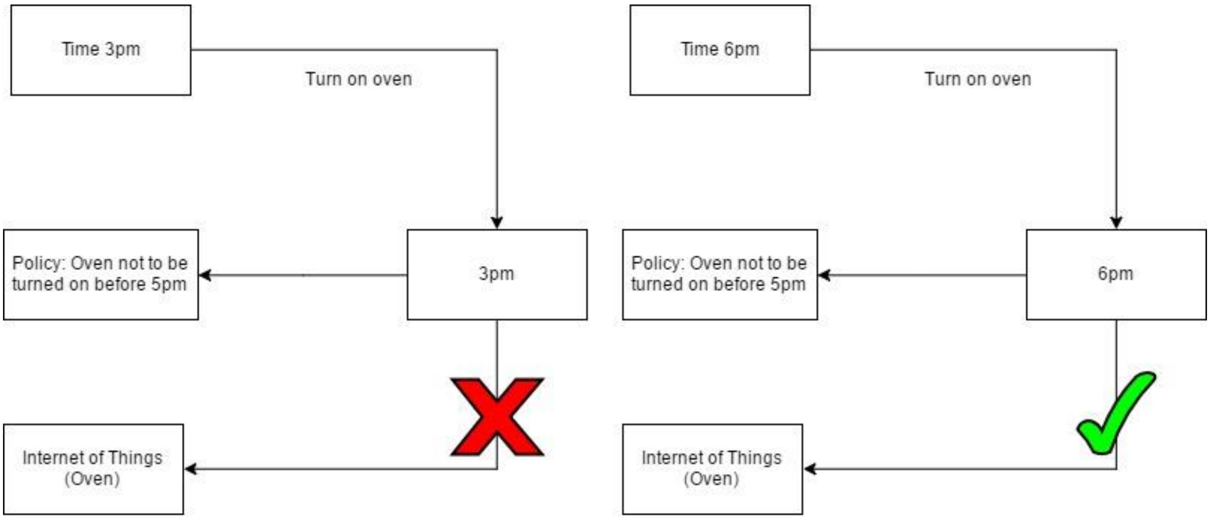

Time 3pm
Turn on oven
Policy: Oven not to be turned on before 5pm
3pm
Internet of Things (Oven)
Time 6pm
Turn on oven
Policy: Oven not to be turned on before 5pm
6pm
Internet of Things (Oven)

FIGURE 1.1 – EXAMPLE OF HOW POLICY ENGINE WILL WORK

That is just one example of how the policy engine sat on the server could be used. The idea is that the policy engine will prevent unauthorised access to devices on the network as the user recognises that the trigger is outside the “rule-set”. Thus providing a practical impact on cyber security as it prevents frustration and annoyance for the end user as their devices are not left on or controlled by an attacker, as well as a virtual impact due to denying unauthorised access to the user's network. The solution that I intend to create could be used in office scenarios as well as home scenarios. Another scenario could be an office ran Twitter account which highlights positivity and displays tweets which show positivity and results of the company. In this case, the policy could be a simple profanity filter that checks the tweets before posting them onto the office Twitter account, making sure there is no profanity within those tweets, which could potentially harm the company's reputation.

Establishing the policy in place and making it work on the server will be the most challenging aspect of the project. Working within such an open and diverse market means that the options for policy criteria are vast and the ability to take advantage of these platforms is considerable. Ensuring that the policies are only accessible to the user who created them is essential, as you do not want outside influence over the policies as that would remove the benefit of them. By utilising the range of services provided by IFTTT and highlighting the two largest risk areas, i.e. origin and time, to fully test and establish the usability of the solution. Also look to overcome these challenges by providing a set of rules and guidelines to assist in the filtration of content. With the creation of a layer of security made achievable a clear set of rules and guidelines that will provide users with the verification of the source of the event. The policy will be editable by users only and will detail criteria to allow for automated applications to run in accordance with user's specification.

2 Background

This section will be discussing the current problem as well as the current solutions already available. Throughoutthis section the theory used by these automation applications will be commented on, as well as detail on the tools and languages that will be used throughout this project and why they have been chosen.

2.1 The Problem

The Internet of Things is becoming more and more prevalent in everyday life. Smart devices are being used within our homes, cars and places of work. Smart wearables such as watches and jewellery, Fit Bits or sensors are being used to capture large amounts of data which can be used to help monitor the user's health and well-being. According to Forbes, the number of devices that will be connected to the Internet of Things by the year 2025 is 75.44 billion. [1]

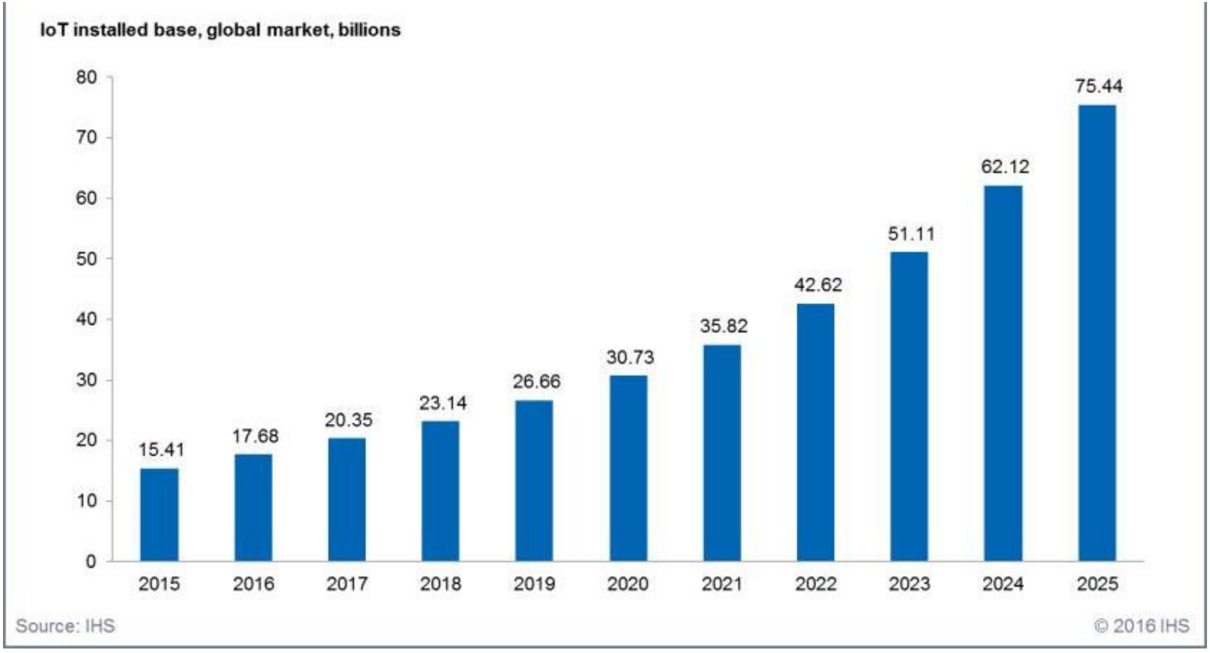


FIGURE 2.1 – PREDICTED AMOUNT OF DEVICES CONNECTED TO IOT BY 2025

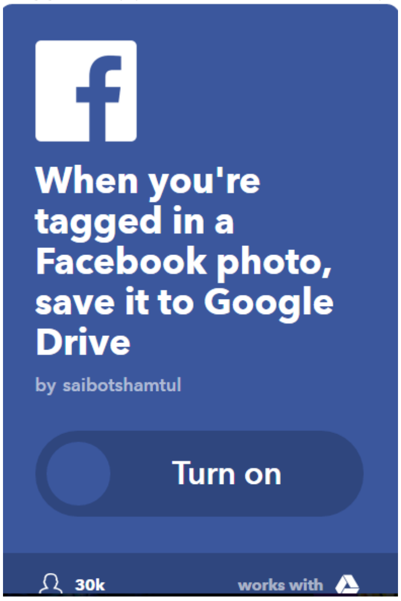
The security of these current devices causes concerns as it is already sub-optimal and as the number of devices being connected to the internet increases, (from lights, cameras, central heating and microwaves) the security of these devices is being brought under scrutiny as companies are often inexperienced in the world of cyber security yet produce fully functioning networked computers in the shape of these smart devices. This is not a concern for these companies as the average user is not worried with the security implications of these devices, Alex Drozhzhin, a cyber security expert, says that even though a microwave is connected to the internet, the average user still believes that it is just a microwave, when in fact it is a fully networked computer [2] which can be used to gain access to your household devices on the network.

As more devices are added and implemented into the IoT, the number of points that a hacker can gain entry is increasing with each device that is added to your home network. [3] Areas of concern within the IoT are: its vulnerability to being hacked, the perception of it by the public and the true security of The Internet of Things. [4] Thus the problem that needs to be addressed is the lack of sufficient security to protect these smart devices, especially ones in the home which can give a hacker potentially easy access to a private and sensitive network where a should feel safe and secure regardless of the device or platform being used on the network.

Automation applications are being widely implemented to make interactions with IoT devices simpler and more convenient for the user. These applications provide an interface for a user to control their networked smart devices through the use of triggers and events. The use of these applications is becoming more frequent as the IoT becomes more easily accessible with the number of devices a user can have. These devices can be anything from door bells, lights in your home, central heating devices such as HIVE, as well as your social media accounts, which can be linked to these applications to help make repetitive and tedious tasks on these platforms less of a task by using automation.
These applications do make tasks easier and help to control a lot of monotonous day-to -day occurrences, however they do not have sufficient security in place to keep the data and devices of their user bases secured. I will be looking into these applications to see what they do and what they could be doing, focusing solely on If-This-Then-That, also known as IFTTT but with consideration to other competitors on the market.

2.2 IF-THIS-THEN-THAT

If-This-Then-That (IFTTT) is a web based application and was introduced in 2010 [5]. The concept of IFTTT is simple, take two services or platforms such as Facebook and Google Drive, and connect these together in order to make repetitive tasks simple and automated. The application creates conditional statements between these two services, one is the trigger or "If-This part", and the second service is the action or "Then-This" part. This is put together to create what IFTTT calls an "Applet", applets consist of a trigger and a corresponding event which occurs after the condition for the trigger has been met. An example of an applet can be seen below, the trigger for this applet to run is when your Facebook account is tagged in a photo, the event which then occurs after this trigger happens, is to save that photo you have been tagged in to your Google Drive account.



When you're tagged in a Facebook photo, save it to Google Drive

by saibotshamtul

Figure 2.2 – Example applet from IFTTT

IFTTT has a simple concept and is quickly becoming more and more popular as more services are added to the application. As more and more devices become available in The IoT, applications such as IFTTT, will only grow in popularity as they offer the services which allow users to automate these devices and give them the tools to control and monitor all of their social media feeds. It makes advertising on social media platforms extremely easy as you can post the same message each day on all of the platforms using a couple of applets without having to manually log in and post the same message each time on the different platforms. The reason IFTTT is so popular and continues to grow in popularity is because it offers the most services out of all the applications on the market (the services are the platforms such as Facebook, Instagram and Dropbox). IFTTT currently has 422 services available on their website whereas a close competitor (Stringify) only has 78 services available. Therefore you can see that IFTTT offers a huge range of services, linking with the overall increase in user appetite for IoT devices.

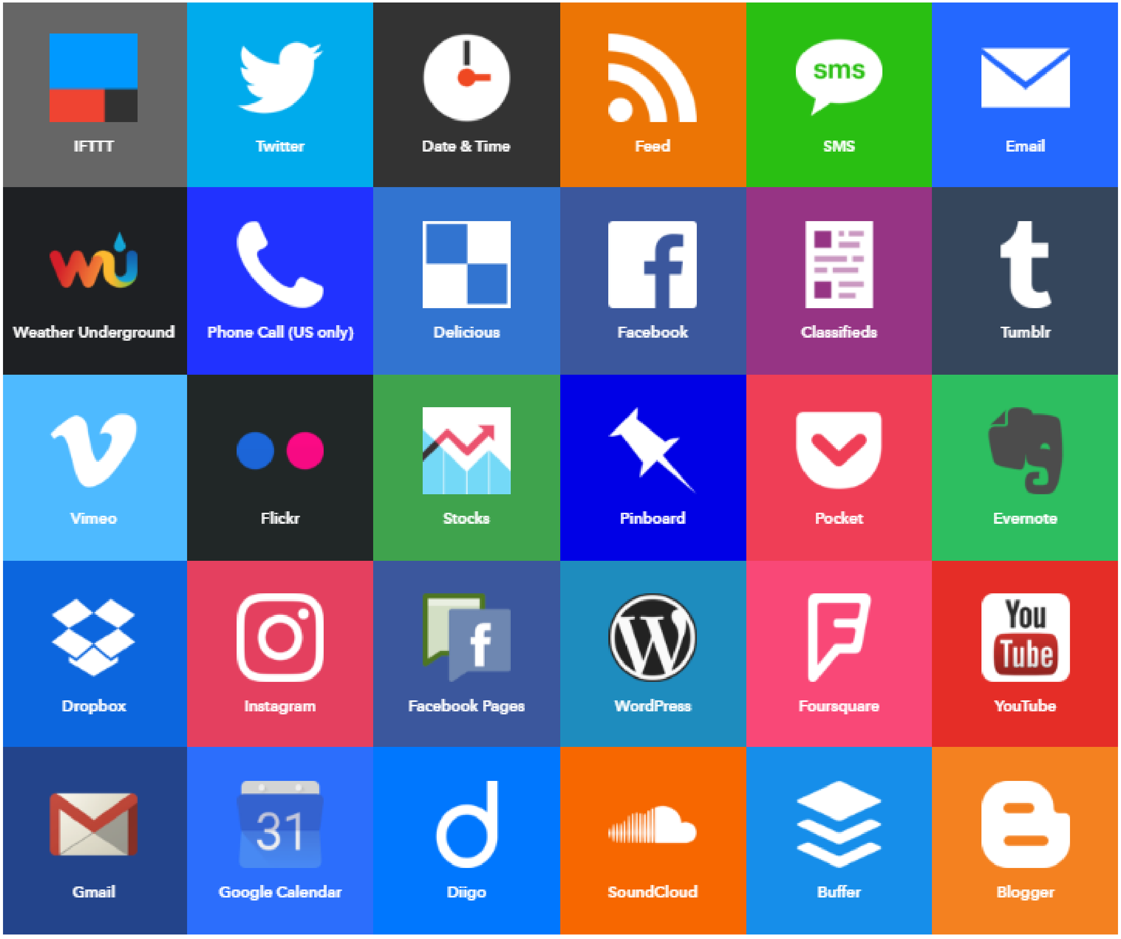


FIGURE 2.3 – EXAMPLES OF THE SERVICES OFFERED BY IFTTT

There are undoubtedly many benefits when using IFTTT and incorporating it into everyday IoT usage, however, there are potential areas of vulnerability. In relation to security, IFTTT uses Secure Socket Layer (SSL) to encrypt information transmitted on their website which is an industry approved way of protecting the information that is transmitted as it is encrypted. In order to use IFTTT on your mobile phone you must give it a lot of permissions such as camera control, contact details, location data, SMS control, storage and telephone, these are just the permissions it needs on your mobile device. In order to use services you must give them access and control over those as well, if you wish to use Facebook you must give IFTTT access to that service so it can post to it and use information on your Facebook feed to trigger events. IFTTT does not store passwords or login details for these third-party services connected via IFTTT although they do store access tokens in a single database. It uses OAuth access as well as access tokens to grant the ability to login and use third-party applications without having to sign in using those credentials each time. Cyber security consultant, Vladimir Jirasek summarises that IFTTT stores OAuth access and refresh tokens in their database which could make them vulnerable to being targeted by hackers. The hacker could gain access to this one database, giving them control over individual services (e.g. Dropbox, Facebook) rather than having to hack each service.[6] Thus if a hacker were to gain access to this database they would have access to all the platforms and services that a user has signed up for using the IFTTT application.

Another concern with IFTTT is that there is no way to verify or control triggers before they occur. A concern shared and expressed by Nitesh Dhanjani, a known cyber security researcher, when he talks of how an attacker can cause a blackout in a user's home by using IFTTT. The applet used in the example is if a user if tagged in a Facebook photo, then change the Philips Hue lights to reflect the colours of the photo. The victim is tagged in a photo which is all black thus the hue lights would reflect the all black colour of the photo and plunge the user's home into darkness. [7] A real world example of how attackers can exploit these applications like IFTTT to manipulate devices within a user's home. This project aims to identify and present with a solution to some of these vulnerabilities.

2.3 Theory Associated with Problem Area

These applications use a lot of terms that could be deemed application specific jargon therefore context has been provided to allow for clarity throughout the piece. [8]

An Applet is a small application that carries out a limited number of simple functions. In the case of IFTTT, an applet consists of one trigger and one action, bringing the services together.

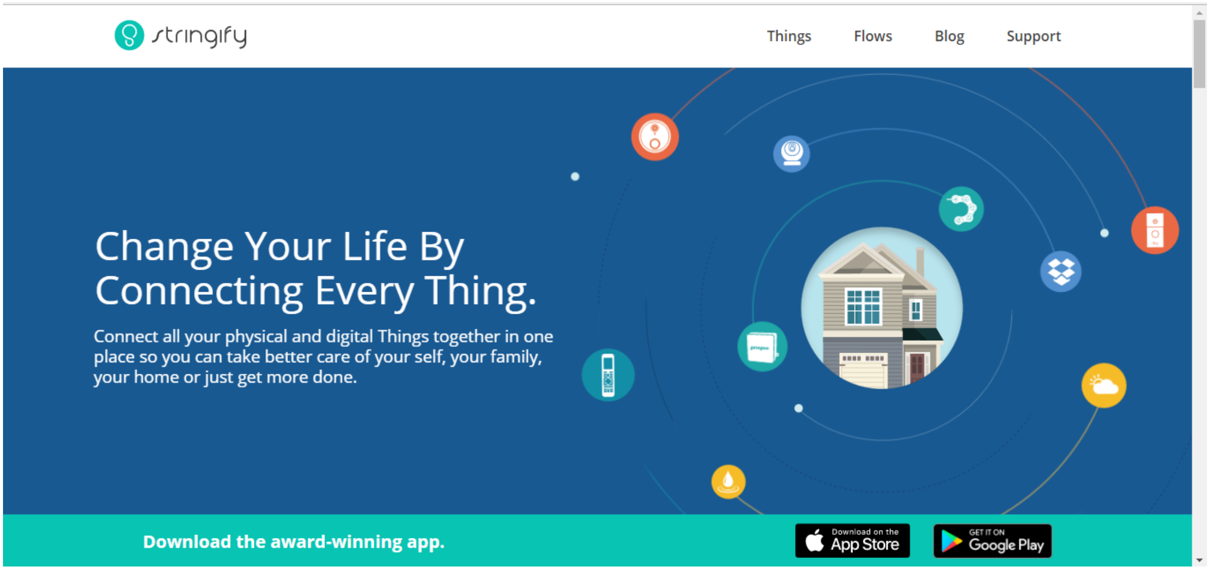
A trigger, in the case of IFTTT, is the "this" part of the application. These items are used to carry out an action, or "trigger" an action.

Services are the apps and devices that you use every day, for example Facebook, Hive, Phillips Hue Lights. There are certain triggers and actions for each service. These are the main building blocks of IFTTT and are used to describe the data from web services.

Actions are the "that" part of the application, for example changing the colour of the Phillips Hue Light. They are the output that results from the trigger.

2.4 Alternative Market Solutions

Aforementioned, there are alternate solutions available to assist in autonomous application, with one such example being Stringify. Stringify works similarly to IFTTT, but rather than just one trigger and one action, it allows multiple parameters for one trigger to set off an action or single parameter triggers to set off multiple actions. This allows a user to have more freedom and customisation when it comes to setting these flows or applets.



stringify

Things Flows Blog Support

Change Your Life By Connecting Every Thing.

Connect all your physical and digital Things together in one place so you can take better care of your self, your family, your home or just get more done.

Download the award-winning app.

FIGURE 2.4 – STRINGIFY HOME PAGE

The security prospects of Stringify are very similar in comparison with IFTTT. Stringify does not store or see login credentials for any third party services or platforms used via their application. [9] It also uses SSL to encrypt sensitive information upon user entry. The application does store access tokens, just like IFTTT, in order to allow a user to use third party services uninterrupted, according to a cyber security analyst, Rose Thibodeaux, the tokens that Stringify use are encrypted whilst transmitted and stored. [10]

The strength of Stringify is that it does provide users with multiple triggers which appears to be a service highly in demand by IFTTT users. However, a key weakness of Stringify is that the number of services offered are limited in comparison with IFTTT's number of services, making IFTTT the preferable option for connecting many different services and devices and thus linking to the increasing market demand of IoT devices. The security weakness of this application is similar to IFTTT, as they have a single database that stores access tokens to all of the services that a user uses on the application, making it the most likely target for an attacker.

Similarly, Stringify does not incorporate any policies that allow control over how or when a trigger is initiated. In practice this would result in an action taking place subsequent to a trigger event regardless of intent or instruction, therefore could jeopardise the security of the IoT device. Given this security vulnerability end users could be susceptible to similar attacks as described in the Phillips Hue Lights example above. A gap which I wish to bridge in this project as the solution will offer a way in which a user can create policies that allow for more control over actions or events occurring.

Another competitor in the market is Microsoft's Microsoft Flow. Microsoft Flow creates flows between apps and services, much like Stringify, allowing multiple triggers and actions for each flow. It has a limited number of services when compared to IFTTT, mainly uses Microsoft products as one would expect, as well as other popular services like Dropbox, Twitter and Facebook. Microsoft Flow is more applicable to businesses trying to connect devices to the IoT due to a feature called the Admin Centre and extra managerial features.

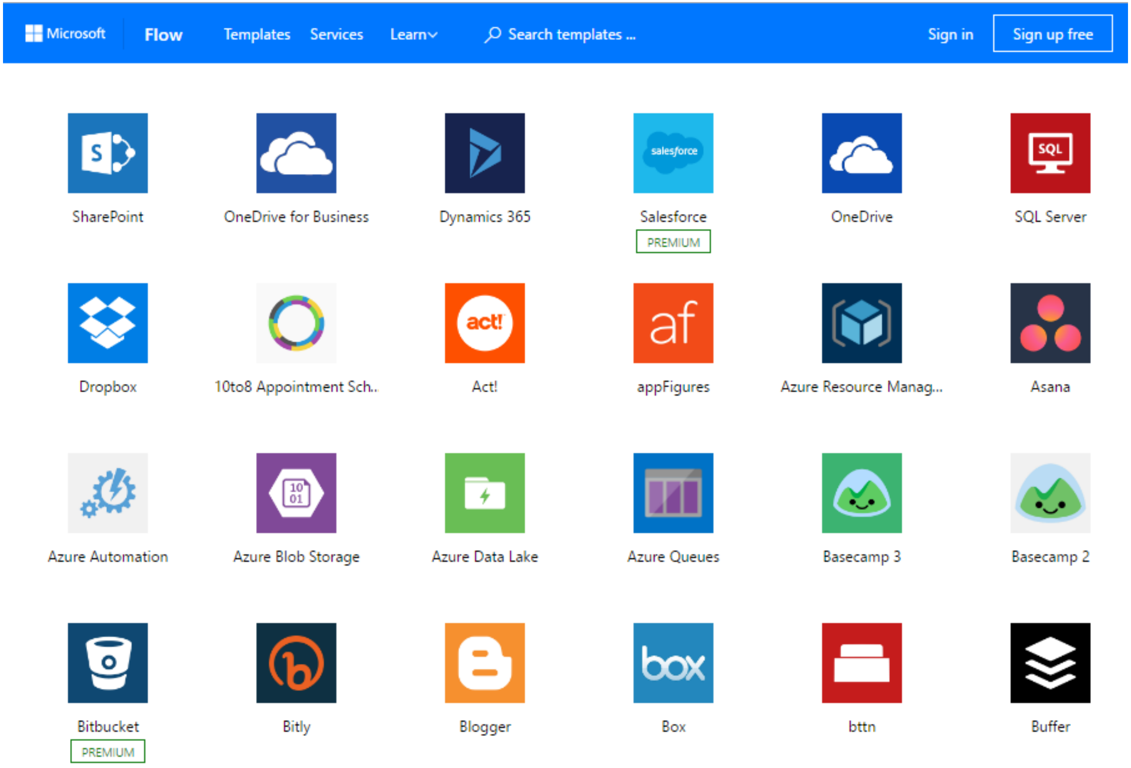


FIGURE 2.5 – MICROSOFT FLOW SERVICES AVAILABLE

Microsoft Flow offers extensive security features as it uses the same security and privacy settings as it does for other Microsoft services and products, such as Microsoft Outlook. These controls include in-depth customer controls within the service, built-in security and best practices [11], more of which can be found on Microsoft Flow's website. A security feature that Microsoft Flow offers that IFTTT and Stringify do not, is the Admin Centre. The Admin Centre allows a user to create an environment, manage permissions and set up Data Loss Prevention (DLP) Policies. [12] DLP policies are Microsoft's way of making sure that business data is not accidentally posted or uploaded onto services like social media platforms, as this data is confidential and is for the business only. For example, preventing data stored in SharePoint being automatically published to a Twitter feed because a trigger occurs which points to this action.

Therefore, a benefit of using Microsoft Flow is the security they offer for adherence to the data protection and the prevention of data loss. The DLPs are an effective way to help set up policies in order to monitor and keep certain information away from actions and events of these "Flows" to keep it secure. However, a material drawback of this application is the range of services on offer in comparison to IFTTT. The standard DLP template that Microsoft and other businesses deploy, have influenced the enhanced layer of security detailed within this project to give users more control over the applets themselves, dependent on trigger event. The policies recommended throughout this piece will augment the existing DLP policies in the market.

2.5 Tools

For the proposed solution, the main programming language that will be used is Python. The functionality of Python is appropriate and efficient allowing me to implement the solution. It also corresponds with the Python micro-framework Flask, as well as the Raspberry Pi, making it the clear choice over other possible languages such as Java. Python is also the main programming language used Debian Operating system. Python has built in libraries that I will be using to control the Raspberry Pi General Purpose Input/outputs (GPIO). Thus clearly being able to see a visual prompt as to how the solution is progressing.

The server software that I will be using is a micro framework for Python named Flask. Flask is a suitable choice for my solution as it is simple to setup, comes with a built-in debugger which offers support and shows the errors that occur on the webpage itself without forcefully exiting the server application. It does not need any configuration or setup before using it. Configuration was given to the usage of Apache as the server, however, the ease of running Python scripts on the Flask server made it the more suitable choice.

Another tool that will be used for the solution is a Raspberry Pi model 3. I have decided to use a Raspberry Pi to visually demonstrate rule sets and policies. The Raspberry Pi is a suitable choice of hardware, as it interacts well with the Debian operating system which is used on the Raspberry Pi. I believe that the Raspberry Pi will help to demonstrate and provide a visual aspect that would not have otherwise been available. An alternative could have been an Arduino board. But due to the compatibility of the Raspberry Pi with the Python programming language, the Arduino was not investigated any further.

I am planning on using Dweet.io in order to communicate between IFTTT and the server. Dweet.io is a service which is used by a device to publish and subscribe to data. It uses machine-to-machine communication for the IoT. It is seen as a sort of Twitter but for "things", which is especially useful for devices connected through the Internet of Things. [13] I have chosen to use Dweet.io as it will be useful in terms of sending push notifications or commands from IFTTT to my web server, acting as the intermediary between the two, as for IFTTT to work for the solution I will need a publicly accessible URL, which Dweet.io provides.

Share your thing — like it ain't no thang.

(Hint: For BIG fun, try it on your smartphone or tablet)

Ridiculously simple messaging
(and alerts)
for the Internet of Things.

FIGURE 2.6 – DWEET.IO HOME PAGE

However, there are alternatives that I could use instead of Dweet, such as ThingSpeak, but ThingSpeak [14] offers more complex features such as analytics and Acts which could overcomplicated the solution I look to implement. Dweet.io is also a free to use platform allowing the user to simply use the code available on their website to listen for dweets. Dweet.io can also be imported and easily integrated into Python which is aligned to the premise of the overall solution.

I will be using Putty in order to remotely access the Raspberry Pi from a laptop. Putty will allow me to login and remotely control the Raspberry Pi without having to attach a mouse, keyboard or monitor to access the Pi on its own. This makes interacting and using the Pi much easier and will allow me to work on both the laptop and the Pi at the same time.

3 The Approach

In this section I will be discussing the problem that my solution will be overcoming, giving an overview of my solution and detailing each different aspect and how I came to this solution. Furthermore, I will include flow diagrams and use cases to demonstrate how the system will interact with users.

3.1 Formal Problem

Currently, there is no security layer on IFTTT that allows a user to customise and configure how these applets are executed. As described above by Dhanjani, an attacker could take advantage of a user who uses an applet to trigger their Phillips Hue Lights when tagged in a Facebook photo. This does seem like a rare situation, but other attacks could be carried out that would trigger applets in a way which the user may not have intended, such as protecting reputation of firms and device security at home. Therefore the solution to this problem is to introduce a security layer that allows a user to configure their applets by putting in place policies and rules to control how their applets are executed. An example of a policy that could prevent this type of attack would be that if the user who has tagged you in a photo is not on your Facebook friends list, then do not execute the corresponding action for that applet.

3.2 Solution Overview

In order to solve this problem, a lack of a security layer, I am going to implement a web server on a Raspberry Pi, configured with a set of policies, which interacts with IFTTT. The webserver will be listening for commands sent by IFTTT to Dweet.io and when they come through, the server will execute the specific functions in the Python configuration file dependent on the command received through the web request. Once the command is executed from the server, the Raspberry Pi will light up the LEDs attached to it, either green or red depending on successful criteria being met. If they are met, then the green LED will come on, and will execute another applet on IFTTT to send a notification to my phone for example. If the conditions are not met, the red LED will be lit up and a message received in the terminal notifying the user why the conditions have not been met. Figure 3.2 (page 15) shows how the solution will add a security layer to IFTTT and the interactions that will take place. Figure 3.1 displays an overview of the proposed solution and how it interacts with the different components.

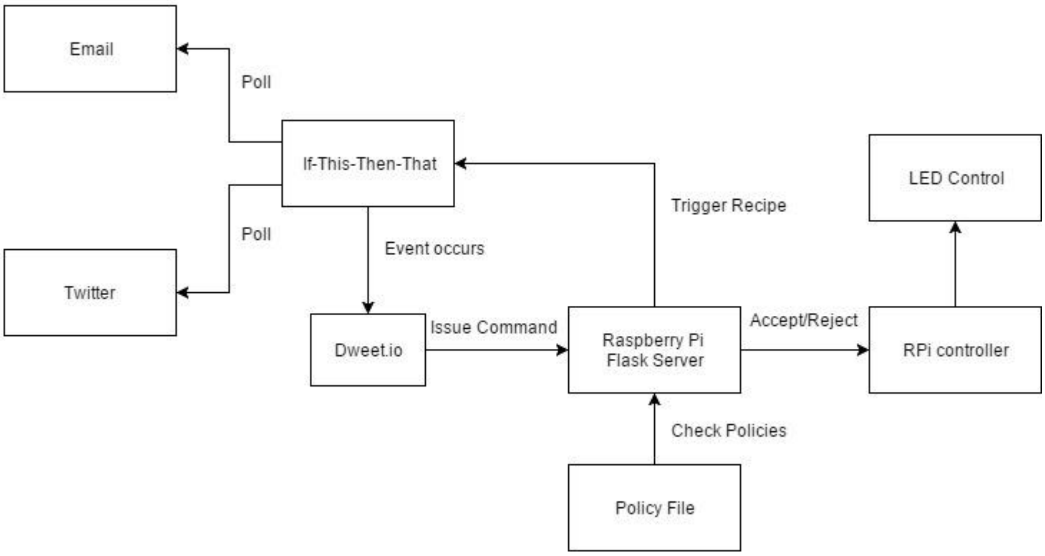


FIGURE 3.1 – SYSTEM OVERVIEW

3.3 Solution Description

The first part of the solution will revolve around a number of applets created on IFTTT that will use Twitter and Email services to demonstrate how a security layer can be implemented. To demonstrate what I can do with IFTTT in order to make it more secure, I will be using the Twitter service as well as the Gmail and Mail365 services. I have chosen to use these services as they allow me to test them myself relatively easily as I can create applets that listen for emails and tweets from my personal accounts, allowing for suitable testing to be carried out. I came to this decision as from my experience with IFTTT, Twitter and Gmail provided a fast response in terms of applet execution, as well as being two of the services available that allow a user to trigger a web request to a publicly accessible URL which is what is needed to communicate between IFTTT and a Raspberry Pi.

The solution would not be possible on other automation applications, and is only viable for IFTTT due to the Maker service that is available on IFTTT. The Maker service allows a user to connect IFTTT applets to networked devices such as Arduino and the Raspberry Pi [15], this will allow the networked Raspberry Pi sat on the network to respond to the applets that are triggered on IFTTT by tweets or emails. The Maker service will allow web requests to be sent from IFTTT to Dweet.io in order to pass variables that that will correspond to a configuration file running on the Flask web server.

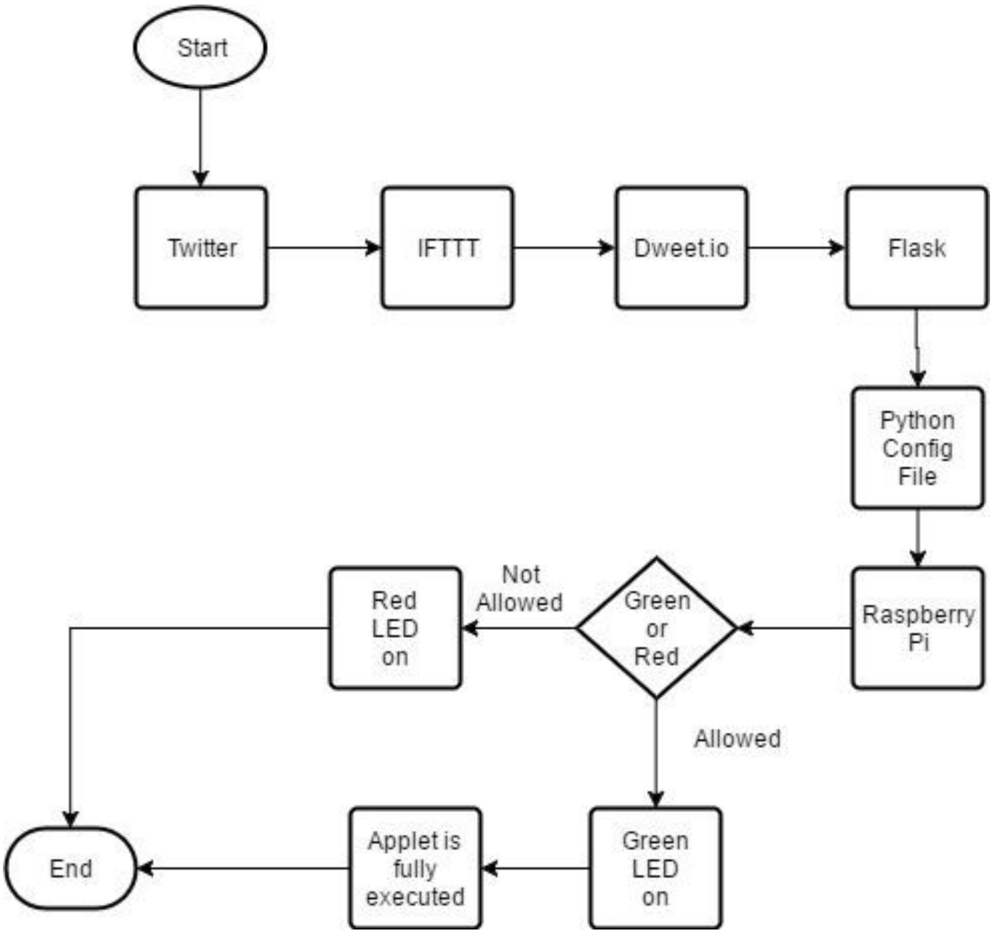


FIGURE 3.2 – LIST OF ALL PROCESSES IN THE INTENDED SOLUTION

Dweet.io is a service that could be used in the solution, and can be used alongside IFTTT to listen for commands. This led to the conclusion that Dweet.io would be useful and viable to use as a part of the intended solution. Dweet.io can be used as a publicly accessible URL required in order to make a web request and thus creating a script which listens for these commands sent from IFTTT through Dweet.io to integrate it with the Raspberry Pi. Dweet.io uses a unique key for example “tweet-test”, so that it knows which channel to listen for commands on. The security of the key relies on it being unique otherwise multiple users with the same key would have control over the commands sent, therefore it will need to be a mix of characters to ensure a unique key is obtained.

For the solution to work online, I will need to setup a web server in order to listen to commands which can then be relayed onto the Raspberry Pi. For this I am going to use Flask, as this will allow me to create a simple web application that can be used as a server. Flask also integrates fully with Python scripts which can be created for interaction between the server and the Raspberry Pi. I intend to setup Flask on the Raspberry Pi itself so that it can be used to control the GPIO pins on the Raspberry Pi.

The solution will need a file to control policies and rules that will be set up on the server to control whether applets are executed or not. This will be the layer of security and the whole basis of the solution. The configuration file will sit on the server and wait until events come in from IFTTT so that it can respond to them, depending on the command sent through to it via Dweet.io. When IFTTT applets are triggered, they will check the policies in this file in order to see whether or not they are accepted by the server in order to execute the follow up applet to trigger a notification. For this solution, the policies in practice will be hardcoded into a Python file that will sit on the Raspberry Pi and be called when an event occurs that needs to be compared against the policies in place. An alternative approach would be to have a text file with policy descriptions and type, rather than encoding the policies directly into the Python code. If an event occurs then the server would compare the event against the policies on the server, checking if the event is of a certain type and how to proceed as a result. In practice, coding the policies into the configuration file is simpler and will be the approach that I am taking as it will provide full control over how the events are compared against the policies without the ambiguity of a text file with multiple policies.

A Raspberry Pi will help to visually demonstrate the effects of the security layer through the use of LEDs attached via a breadboard. The Raspberry Pi will also be used as a web server with Flask running on it, allowing the server to listen for commands from IFTTT and Dweet.io in order to control the GPIO pins on the Raspberry Pi and to turn on the LEDs. If the command is accepted by the policy then the green LED will light up, otherwise the red LED will light up. Some solutions available on the web already use a Raspberry Pi to help act as a web server for connecting itself and IFTTT together, therefore, is relevant and appropriate to this solution.

3.4 Use Case Diagram

Figure 3.3 is showing how the solution will work when a user triggers a recipe that is accepted by the rules and policies hosted on the server. In this use case, a user sends a tweet at 11am, triggering their IFTTT applet which then makes a web request to Dweet.io with the command = Time. The web server then listens on Dweet.io’s service for this command, when it receives the command it checks the configuration file for that specific policy regarding time. The policy in place could be a rule that if a tweet is sent between 9-5pm then it is allowed and thus will turn the green LED on and trigger a second applet on IFTTT to send a notification to the user’s phone saying they have a been mentioned in a tweet or a user has made a new tweet for example.

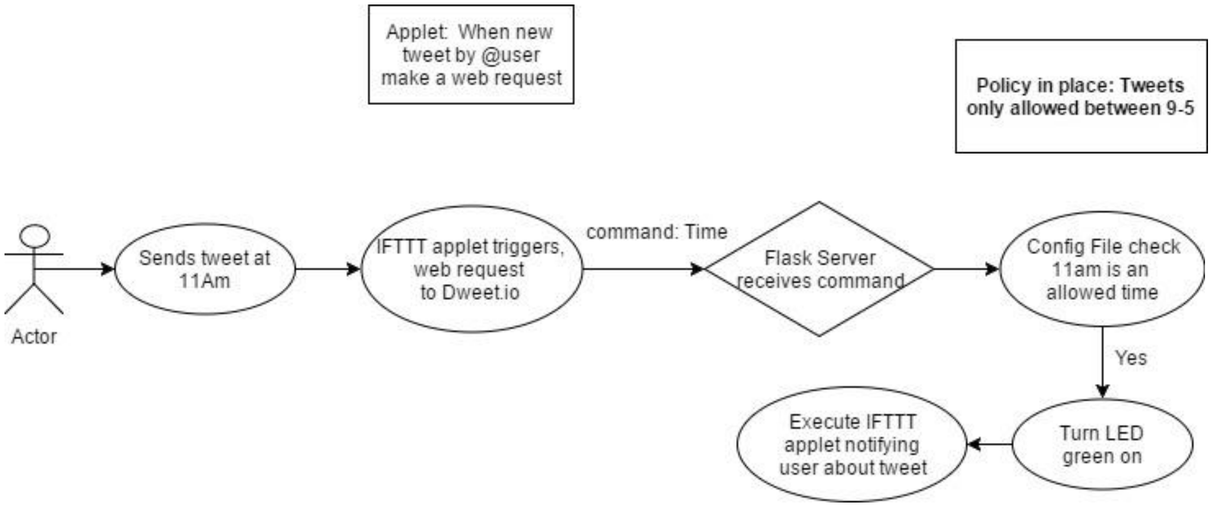

Applet: When new tweet by @user make a web request
Policy in place: Tweets only allowed between 9-5
Actor
Sends tweet at 11Am
IFTTT applet triggers, web request to Dweet.io
command: Time
Flask Server receives command
Config File check 11am is an allowed time
Yes
Turn LED green on
Execute IFTTT applet notifying user about tweet

FIGURE 3.3 – USE CASE DIAGRAM OF USER SENDING A TWEET AT 11AM WHICH IS ALLOWED

Figure 3.4 shows the same scenario but in the case of if the tweet was sent at 1am instead of 11am. The user would send a tweet to trigger the web request. IFTTT would then see a new tweet has been posted by the user or whatever the conditions of the applet are regarding twitter to trigger the web request (it could be a new tweet by a user, a mention of their account or a certain hashtag is used), IFTTT sends the command = Time to Dweet.io, the web server is listening for this command and when it receives it, it will check the configuration file sat on the server to see what the necessary action is for that command. Seeing that the time of the tweet is 1am, and the rule in place says tweets only allowed between 9-5pm, the Raspberry Pi would turn on the red LED not triggering the notification on the user's phone. The time ruling in the policy could be a useful practical benefit as it would prevent disturbances to the user outside of working hours.

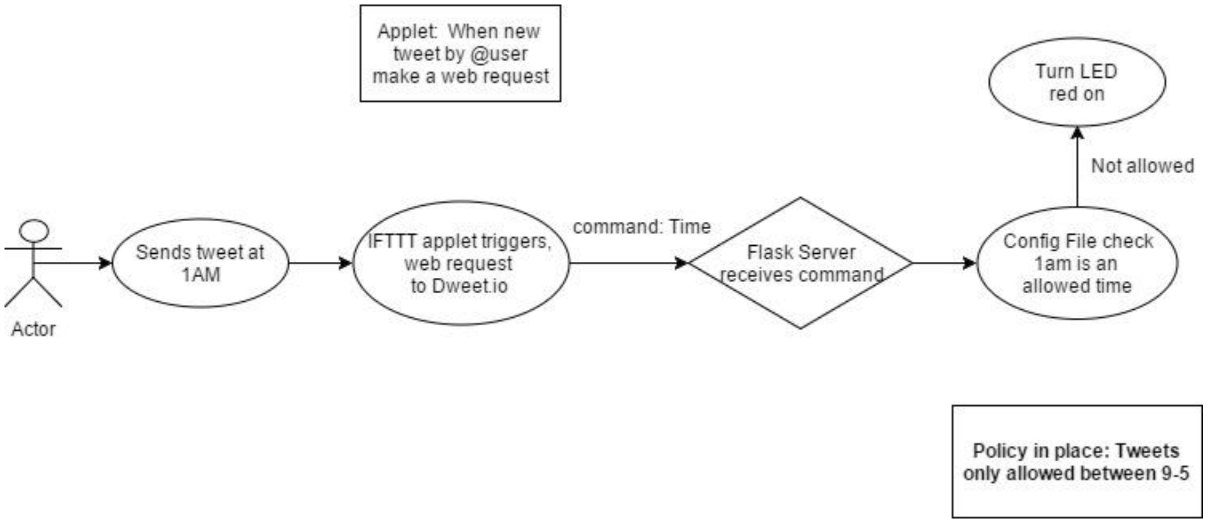

Applet: When new tweet by @user make a web request
Turn LED red on
Not allowed
Actor
Sends tweet at 1AM
IFTTT applet triggers, web request to Dweet.io
command: Time
Flask Server receives command
Config File check 1am is an allowed time
Policy in place: Tweets only allowed between 9-5

FIGURE 3.4 – USE CASE DIAGRAM OF A USER SENDING A TWEET AT 1AM WHICH IS NOT ALLOWED

Another use case would be if the origin of the source is not a trusted or known entity. The intended solution will be able to check whether or not the user who triggered the applet is a trusted user. If the user was not trusted then you could turn the red LED on and say that the origin of the trigger is not trusted. For this solution, I will be using Twitter and emails, therefore if the user account on Twitter is not in a verified user list in the configuration file then the applet will not be executed further. It would work in the same way for emails, if the email address that triggers the applet on IFTTT to run is not a verified source then the applet will not send a notification to the user.

Figure 3.5 shows that @user sends a tweet triggering the applet on IFTTT which sends the web request, passing the command = Origin onto Dweet.io. The Flask server on the Raspberry Pi listens and gets the command = Origin, the server will check the command against the policy in place in the configuration file on the server. The policy in place is that only @user1 and @user2 can trigger the applet, therefore because @user is not a recognised source, the red LED on the Raspberry Pi will light up and a message appears in the terminal explaining why this has happened.

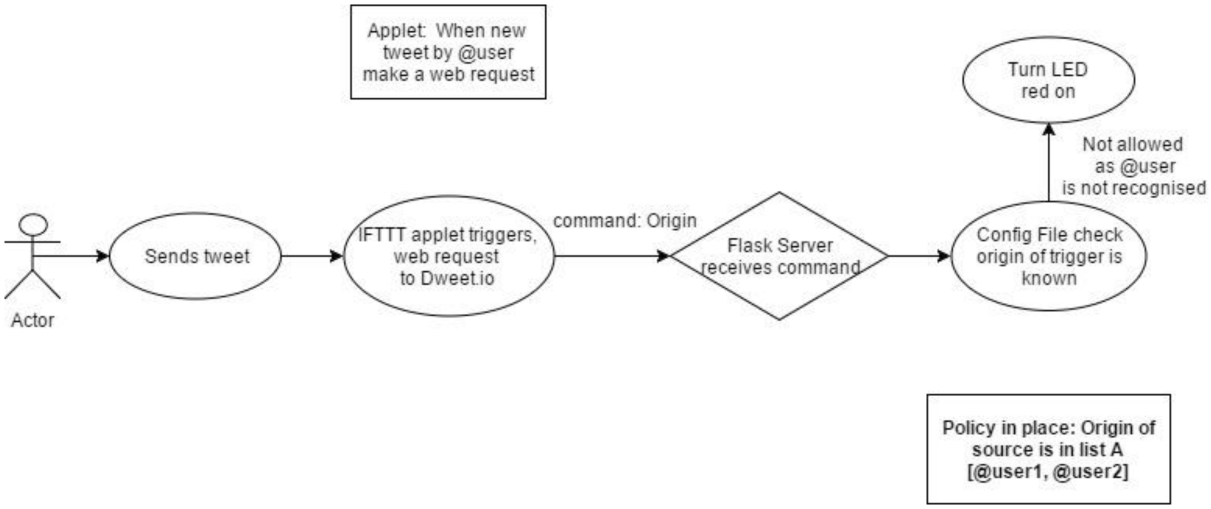

Applet: When new tweet by @user make a web request
Turn LED red on
Not allowed as @user is not recognised
Actor
Sends tweet
IFTTT applet triggers, web request to Dweet.io
command: Origin
Flask Server receives command
Config File check origin of trigger is known
Policy in place: Origin of source is in list A [@user1, @user2]

FIGURE 3.5 – USE CASE DIAGRAM OF USER SENDING A TWEET WHO IS NOT A VERIFIED SOURCE

These are specific examples that I intend to use for the solution in order to demonstrate how the system could be used for almost any applet on IFTTT rather than just Twitter and email.

4 Implementation

This section will be detailing the parts of the project which have been changed in relation to the initial plan due to circumstances and ideas occurring. It will demonstrate sections of code which are key to the solution and which were difficult to create. As well as demonstrating other services utilised for the project.

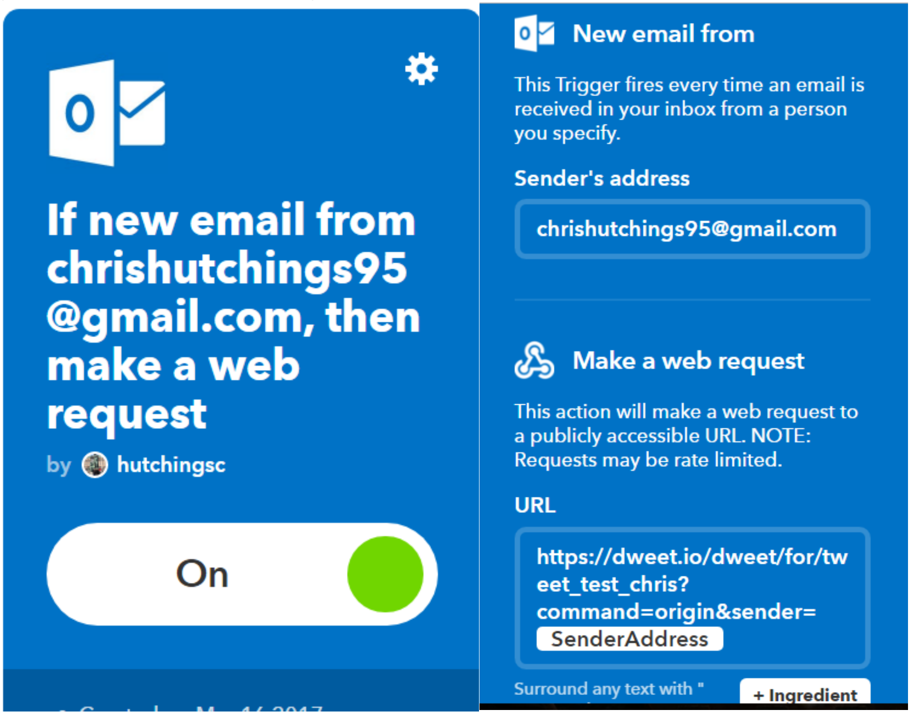
4.1 Initial Plan Changes

At the beginning of the project, the initial idea was to create an environment that was similar to IFTTT and build a security layer on top which would allow users to create their own applets. However, after realising that was not very achievable, and some discussion with fellow academics I came to the conclusion of using a Raspberry Pi. The Raspberry Pi could act as a web server that would communicate with IFTTT and Dweet.io in order to visually demonstrate that the security layer implemented was working and carrying out some actions after receiving commands.

It is also worth noting that certain elements of the security policy previously discussed i.e. location of user and channel encryption were not investigated as part of this solution. The decision was made to focus on realistic and achievable targets, such as time scales and source, and given further research capability, more complex matters such as channel encryption could be investigated at a later date.

4.2 Created IFTTT Applets

As mentioned in the above section, the project is using Twitter and Email services from the IFTTT website. The applet in figure 4.1 uses the email service and states that whenever the email account linked to IFTTT receives an email from chrishutchings95@gmail.com then make a web request to the following URL, https://dweet.io/dweet/for/tweet_test_chris?command=origin&sender={{SenderAddress}}. Thus when an email is received from 'chrishutchings95@gmail.com' it creates a 'dweet' to Dweet.io on the channel 'tweet_test_chris' which is the unique channel I have created. The dweet sends two parameters as key value pairs, these are command=origin and sender=SenderAddress, these parameters can be used by the web server later on.



If new email from chrishutchings95 @gmail.com, then make a web request

by hutchingsc

On

New email from

This Trigger fires every time an email is received in your inbox from a person you specify.

Sender's address

chrishutchings95@gmail.com

Make a web request

This action will make a web request to a publicly accessible URL. NOTE: Requests may be rate limited.

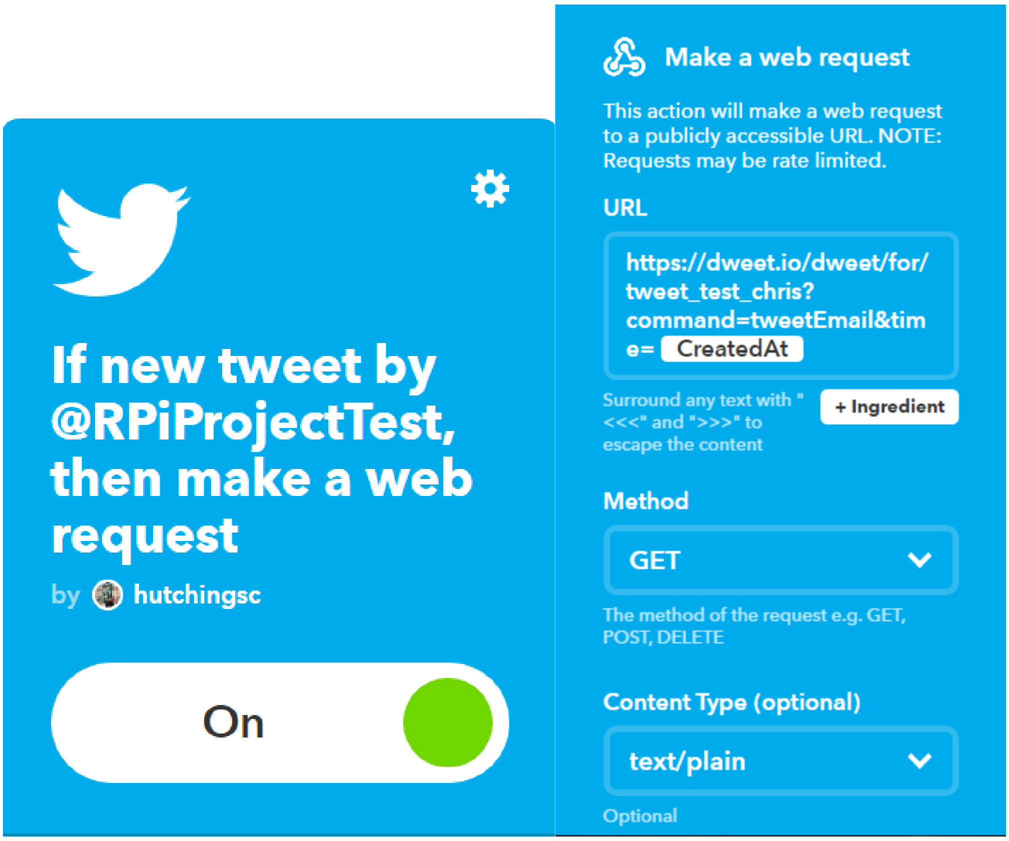
URL

https://dweet.io/dweet/for/tweet_test_chris?command=origin&sender= SenderAddress

Surround any text with "

+ Ingredient

FIGURE 4.1 – IFTTT APPLET WHEN RECEIVING AN EMAIL FROM CHRISHUTCHINGS95 MAKE A WEB REQUEST



If new tweet by @RPiProjectTest, then make a web request

by hutchingsc

Make a web request

This action will make a web request to a publicly accessible URL. NOTE: Requests may be rate limited.

URL

https://dweet.io/dweet/for/tweet_test_chris?command=tweetEmail&time= CreatedAt

Surround any text with "<<<" and ">>>" to escape the content

Method

GET

The method of the request e.g. GET, POST, DELETE

Content Type (optional)

text/plain

Optional

FIGURE 4.2 – IFTTT APPLET FOR NEW TWEET BY @RPIPROJECTTEST, MAKE A WEB REQUEST

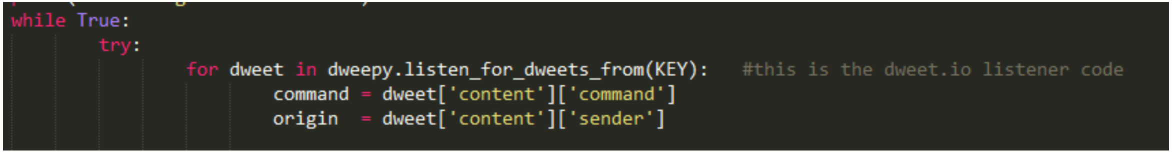
The applet in figure 4.2 is an example of the usage of the Twitter service on IFTTT. This applet is triggered when a new tweet is created on my Twitter account. As soon as the tweet is posted, IFTTT will check the Twitter service and trigger this applet by making a web request to the URL https://dweet.io/dweet/for/tweet_test_chris?command=tweetEmail&time=CreatedAt. This web request works in the same way as the applet in figure 4.1, the URL makes a “dweet” to the channel “tweet_test_chris” with the parameters command=tweetEmail and time=CreatedAt. This command is later used by the web server to execute the specific functions for “tweetEmail” which is located in the configuration file stored on the server.

The parameters sent in both of the applets shown in figure 4.1 and 4.2 can be anything that the user desires. I have chosen to use command and then a parameter that is specific to that applet which helped to identify variables that were needed in order to add in the security layer. As seen in figure 4.2, I sent the parameter time=CreatedAt, the configuration file can use the time sent by IFTTT to check that it is between the set time allowed in the policies on the web server.

4.3 Dweet.io Code Segment

Figure 4.3 shows a segment of code that is used to listen for the "dweets" sent by IFTTT with the parameters. This part of the implementation is critical to the system being able to work, it uses the Python library, dweepy, which is the Dweet.io compatible library for Python. The Dweet.io part of the code was discovered in The Raspberry Pi Cookbook as one of the projects, it works by listening on the unique key provided (KEY = tweet_test_chris). The code listens for any messages from Dweet.io's server that arrives from IFTTT. [16]

The command variable takes the content of the "command" that arrives from IFTTT in the form of a Dweet.io web request, origin variable does the same thing for the "sender" parameter from IFTTT, this gives access to the parameters to be used in the configuration file for the policies to be created and check against these parameters.



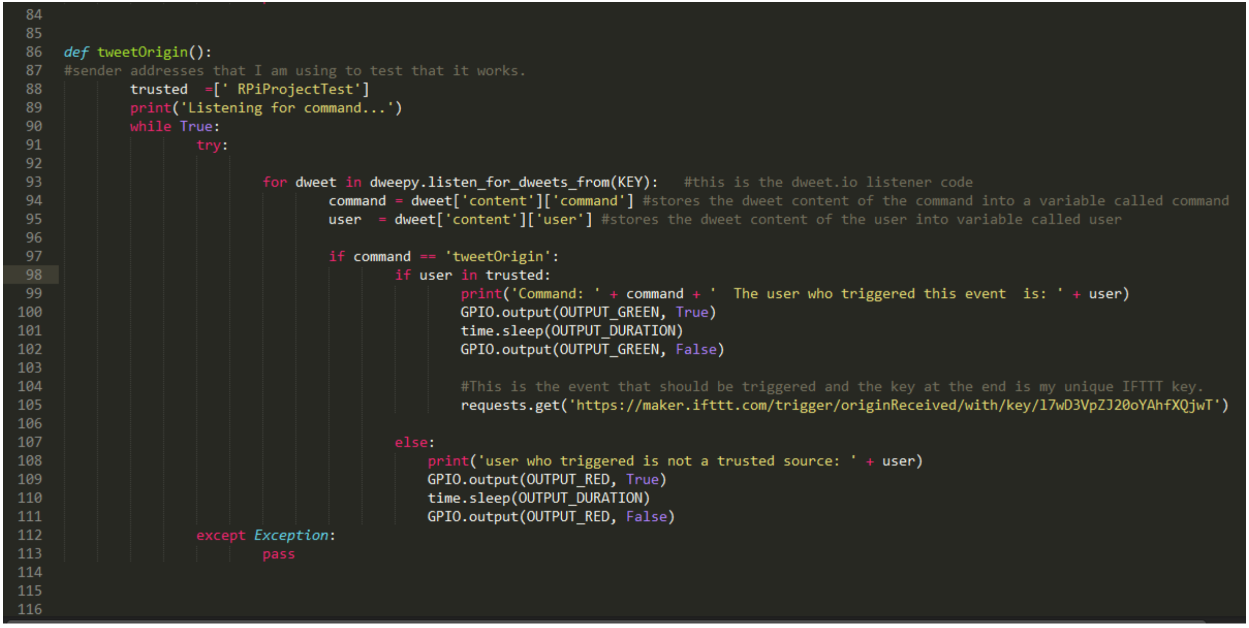
```
while True:
    try:
        for dweet in dweepy.listen_for_dweets_from(KEY):    #this is the dweet.io listener code
            command = dweet['content']['command']
            origin  = dweet['content']['sender']
```

FIGURE 4.3 – DWEETORIGIN CODE SEGMENT

The code displayed in figure 4.3 is used for each policy setup in the configuration file as each section is used to capture certain parameters sent from IFTTT depending on the rule setup for that applet. Without the discovery of this section of code, the project would have been much harder to find a way to connect IFTTT to the Raspberry Pi. Therefore this code segment was useful for this project as it allowed me to retrieve the commands sent from IFTTT easily through the use of the Dweet.io server. I was then able to send the commands to the Raspberry Pi web server that was listening in order to execute the functions in response to the applet executed on IFTTT.

4.4 Configuration File

The configuration file is the most important part of this project as this is the security layer that has been implemented in order to solve the problem. The configuration file sits on the Raspberry Pi and is called by the web server where it is used to control what occurs on the Raspberry Pi itself, as well as whether or not the second notification applet on IFTTT is triggered.

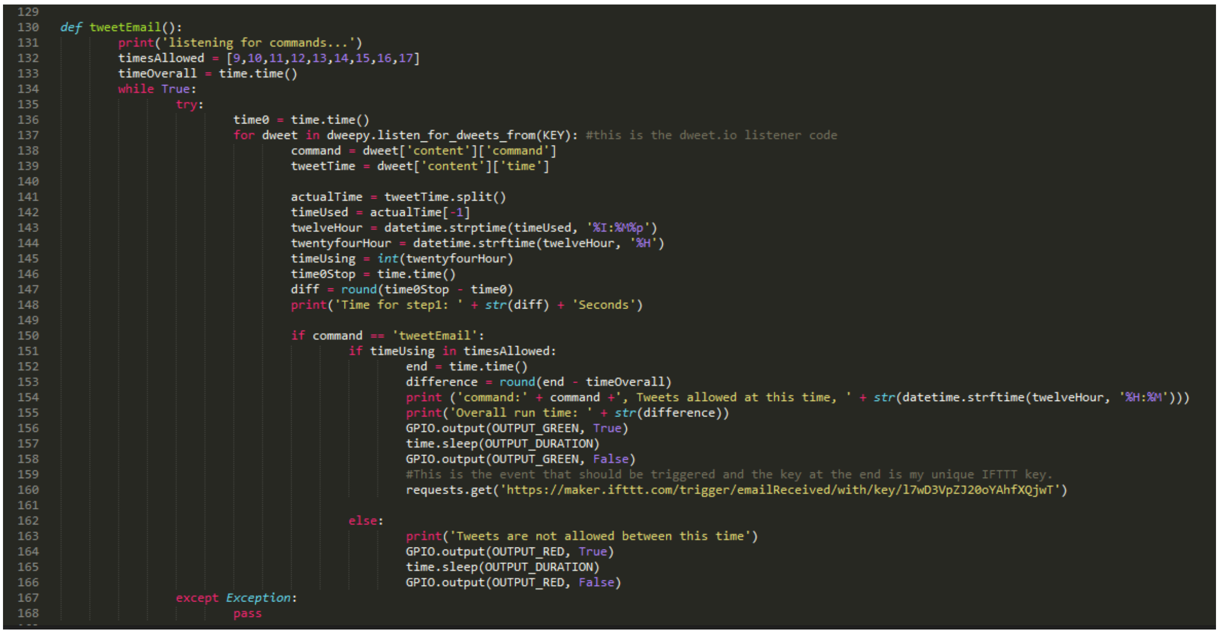


```


def tweetOrigin():
#sender addresses that I am using to test that it works.
        trusted  =[' RPiProjectTest']
        print('Listening for command...')
        while True:
                try:

                        for dweet in dweepy.listen_for_dweets_from(KEY):   #this is the dweet.io listener code
                                command = dweet['content']['command'] #stores the dweet content of the command into a variable called command
                                user  = dweet['content']['user'] #stores the dweet content of the user into variable called user

                                if command == 'tweetOrigin':
                                        if user in trusted:
                                                print('Command: ' + command + '  The user who triggered this event  is: ' + user)
                                                GPIO.output(OUTPUT_GREEN, True)
                                                time.sleep(OUTPUT_DURATION)
                                                GPIO.output(OUTPUT_GREEN, False)

                                                #This is the event that should be triggered and the key at the end is my unique IFTTT key.
                                                requests.get('https://maker.ifttt.com/trigger/originReceived/with/key/l7wD3VpZJ20oYAhfXQjwT')

                                        else:
                                            print('user who triggered is not a trusted source: ' + user)
                                            GPIO.output(OUTPUT_RED, True)
                                            time.sleep(OUTPUT_DURATION)
                                            GPIO.output(OUTPUT_RED, False)
                except Exception:
                        pass



```

FIGURE 4.4 – CONFIGURATION FILE FOR CHECKING ORIGIN OF SOURCE VIA TWITTER

Figure 4.4 shows one policy that I set up in the configuration file to control the origin of the source. The code has a list which contains trusted users, in this case “RPiProjectTest” which was the Twitter account that I had setup in order to test the solution. As that username is in the trusted list of users, if an applet is then triggered by “RPiProjectTest” it is allowed and the Raspberry Pi would light up green whilst sending a request to IFTTT, triggering the notification applet.

The configuration file itself was challenging as a result of delays on the IFTTT website in its current guise, which could be overcome with increased testing and collaboration with the website, in future iterations of this project.



```
def tweetEmail():
        print('listening for commands...')
        timesAllowed = [9,10,11,12,13,14,15,16,17]
        timeOverall = time.time()
        while True:
                try:
                        time0 = time.time()
                        for dweet in dweepy.listen_for_dweets_from(KEY): #this is the dweet.io listener code
                                command = dweet['content']['command']
                                tweetTime = dweet['content']['time']

                                actualTime = tweetTime.split()
                                timeUsed = actualTime[-1]
                                twelveHour = datetime.strptime(timeUsed, '%I:%M%p')
                                twentyfourHour = datetime.strftime(twelveHour, '%H')
                                timeUsing = int(twentyfourHour)
                                time0Stop = time.time()
                                diff = round(time0Stop - time0)
                                print('Time for step1: ' + str(diff) + 'Seconds')

                                if command == 'tweetEmail':
                                        if timeUsing in timesAllowed:
                                                end = time.time()
                                                difference = round(end - timeOverall)
                                                print ('command:' + command +', Tweets allowed at this time, ' + str(datetime.strftime(twelveHour, '%H:%M')))
                                                print('Overall run time: ' + str(difference))
                                                GPIO.output(OUTPUT_GREEN, True)
                                                time.sleep(OUTPUT_DURATION)
                                                GPIO.output(OUTPUT_GREEN, False)
                                                #This is the event that should be triggered and the key at the end is my unique IFTTT key.
                                                requests.get('https://maker.ifttt.com/trigger/emailReceived/with/key/l7wD3VpZJ20oYAhfXQjwT')

                                        else:
                                                print('Tweets are not allowed between this time')
                                                GPIO.output(OUTPUT_RED, True)
                                                time.sleep(OUTPUT_DURATION)
                                                GPIO.output(OUTPUT_RED, False)
                except Exception:
                        pass
```

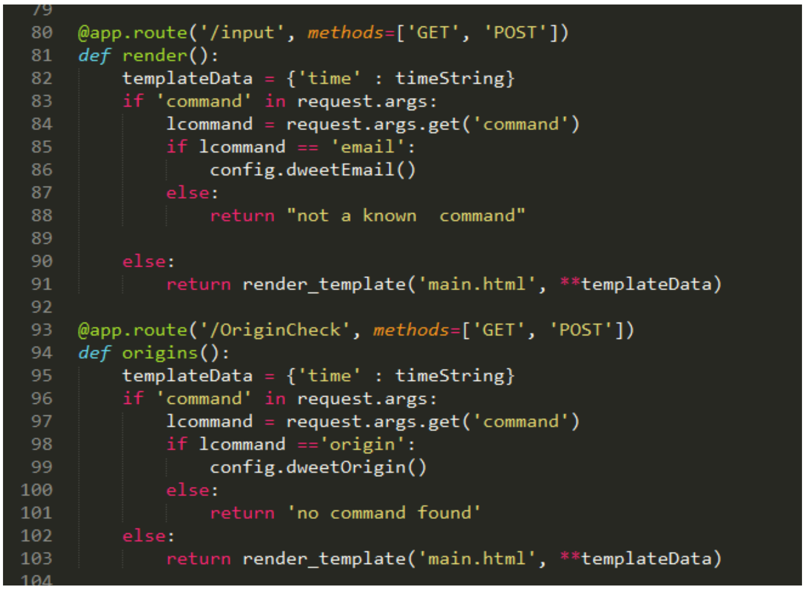
FIGURE 4.5 – CONFIGURATION FILE FOR CHECKING TIMES OF TWEETS POSTED

The code implemented in figure 4.5 was challenging to implement as the timestamp attached to Tweets was in the 12-hour clock format as well as being set to a different time configuration whereas I was using 24-hour clock format, as well as displaying the date of the tweet in the same string. Therefore as seen from the snippet of code in figure 4.5, there are conversions and slicing occurring in order to get the time from the tweet timestamp, then formatting that time to be useable by my configuration file to check it against the policies in place.

4.5 Flask Web Server

The setting up of the web server was a particularly challenging part of the project, due to a lack of experience working with web servers and configuring web servers. At first, I started using Apache as my web server, this was going well until it came to running the Python scripts on it. Due to permission issues and matters arising outside of the scope of the project, including the establishment of a bespoke Linux environment. To overcome the permission issues that I encountered, I carried out research into enabling Python scripts on an Apache web server. As well as listening on ports for commands sent by the client over the connection to the server. I decided this was moving too far away from what the aims and objectives of this project were and therefore decided to try out Flask instead.

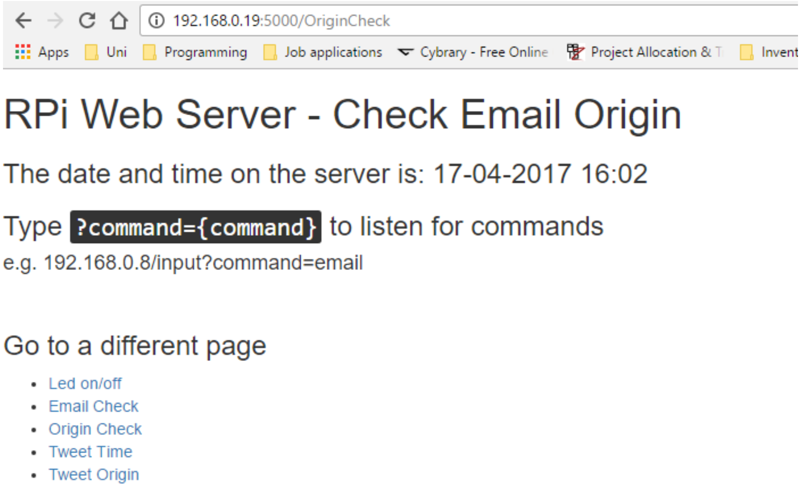
Flask was easier to setup and after a few tutorials and guides, especially Matt Richardson’s basic guide [17], the web server was setup and ready to go. Compared to Apache, Flask being a micro framework for Python was much more Python friendly, allowing Python scripts to be run without any configuration and therefore being much more suitable to the scope of this project.



```
@app.route('/input', methods=['GET', 'POST'])
def render():
    templateData = {'time' : timeString}
    if 'command' in request.args:
        lcommand = request.args.get('command')
        if lcommand == 'email':
            config.dweetEmail()
        else:
            return "not a known  command"

    else:
        return render_template('main.html', **templateData)

@app.route('/OriginCheck', methods=['GET', 'POST'])
def origins():
    templateData = {'time' : timeString}
    if 'command' in request.args:
        lcommand = request.args.get('command')
        if lcommand =='origin':
            config.dweetOrigin()
        else:
            return 'no command found'
    else:
        return render_template('main.html', **templateData)
```

FIGURE 4.6 – TWO PAGES OF THE FLASK WEB SERVER

Once the server was setup and running, I then linked the server up with the configuration file so that the server could communicate the commands sent from IFTTT and Dweet.io to the Raspberry Pi.



RPi Web Server - Check Email Origin

The date and time on the server is: 17-04-2017 16:02

Type `?command={command}` to listen for commands

e.g. 192.168.0.8/input?command=email

Go to a different page

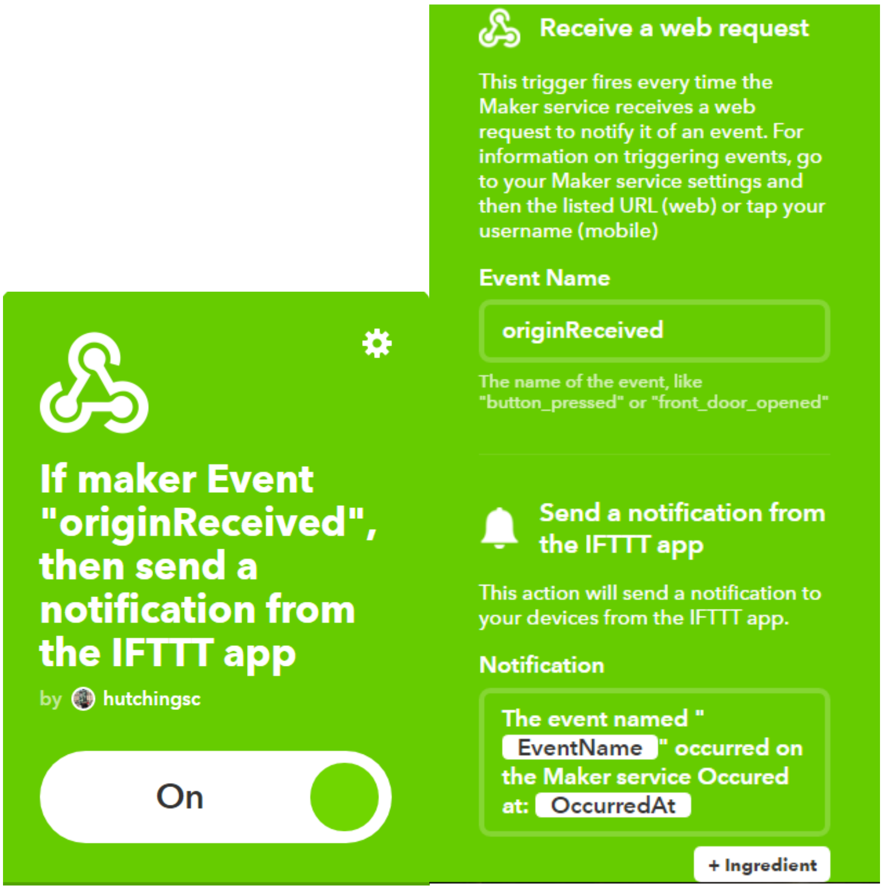
- Led on/off
- Email Check
- Origin Check
- Tweet Time
- Tweet Origin

FIGURE 4.7 – WEB SERVER RUNNING DISPLAYING EMAIL ORIGIN PAGE

The web pages themselves on the server are intentionally straightforward, a template is used to give consistency to each page with a few alterations depending on the page they are on. In figure 4.7, demonstrates the page used for checking the origin of emails to see if the sender of the email which triggered the IFTTT applet is a trusted source.

4.6 IFTTT Notification Applet

The aim of this project was to create a security layer for the IFTTT application. In order to simulate this, I decided to use two separate applets which would eventually be linked to demonstrate what the security layer could be capable of.



If maker Event "originReceived", then send a notification from the IFTTT app

by hutchingsc

On

Receive a web request

This trigger fires every time the Maker service receives a web request to notify it of an event. For information on triggering events, go to your Maker service settings and then the listed URL (web) or tap your username (mobile)

Event Name

originReceived

The name of the event, like "button_pressed" or "front_door_opened"

Send a notification from the IFTTT app

This action will send a notification to your devices from the IFTTT app.

Notification

The event named " EventName " occurred on the Maker service Occured at: OccurredAt

+ Ingredient

FIGURE 4.8 – NOTIFICATION APPLET THAT IS TRIGGERED AFTER SUCCESSFULLY PASSING POLICIES

The way this works is that if an applet is successfully executed and gets through the web server by meeting the policy criteria for that applet, then the applet in Figure 4.8 will be triggered via a get request to IFTTT's website with the following line:
requests.get('https://maker.ifttt.com/trigger/originReceived/with/key/l7wD3VpZJ20oYAhfXQjwT').
Aforementioned, this line triggers the applet in Figure 4.8 by sending a request to IFTTT itself, using "OriginReceived" as the event name and using the unique Maker channel API key to trigger this applet.

4.7 Complete Solution

The complete solution consists of the combination of multiple parts. A Raspberry Pi, a web server running on the Raspberry Pi, the IFTTT application with multiple applets created, Dweet.io and a Python file containing policies. The Raspberry Pi has a breadboard attached to it with two LEDs connected, a green LED and a red LED. For ease of use, the Raspberry Pi is remotely accessed via Putty.

After the Raspberry Pi is powered and connected, the Flask web server was run from the terminal. With the Flask web server up and running, I navigated to the web page for the server and went to the appropriate page, for example Email Check. On this webpage, portrayed in Figure 4.10, the corresponding command was entered. As I was on the Email Check webpage the command to be entered into the URL bar was email. This can be seen in Figure 4.10 at the top of the image in the URL bar.

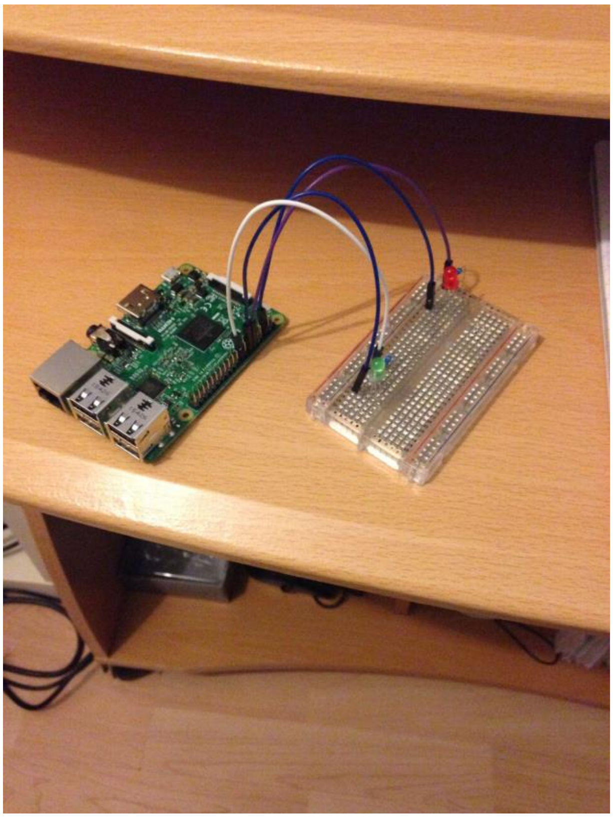
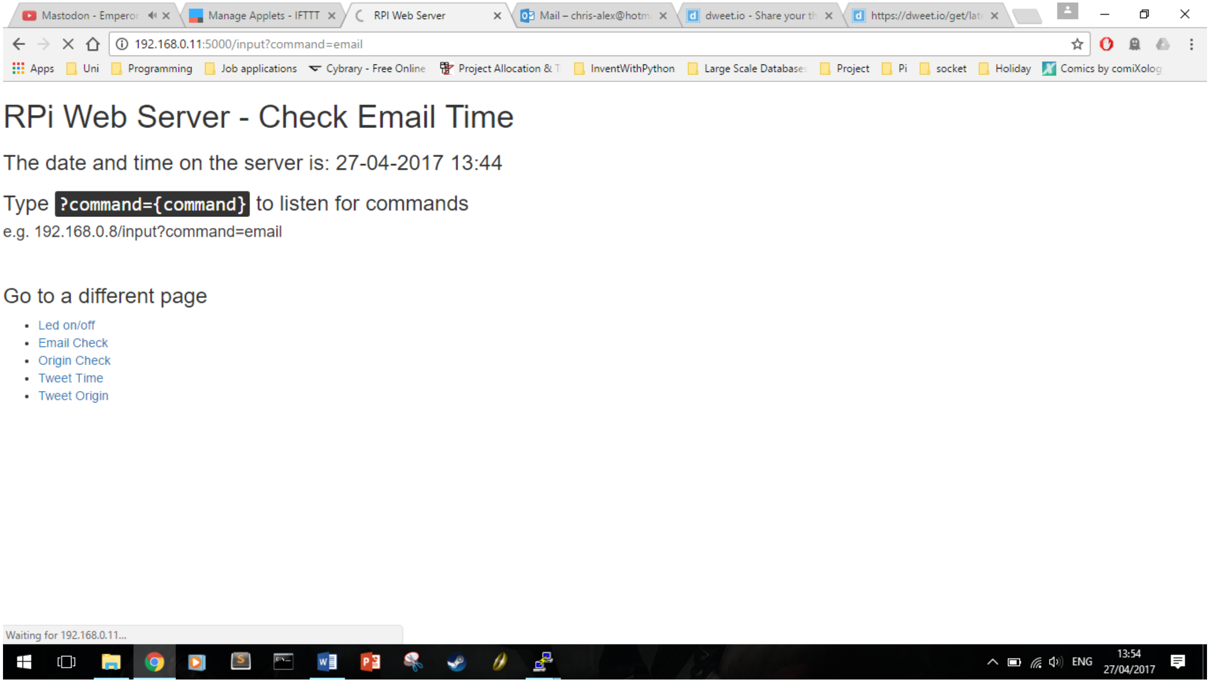


FIGURE 4.9 – RASPBERRY PI WITH BREADBOARD AND LEDS ATTACHED

After entering the command into the URL bar, a message appeared in the terminal saying "Listening for commands…" meaning the Dweet real time subscription code has been activated and the server was waiting for a command to arrive on Dweet.io. In order to get the command to be sent to Dweet.io, the IFTTT email applet must be triggered. This was accomplished by sending an email from a Hotmail address to the linked email address on IFTTT. When the applet was triggered, a web request was sent to Dweet.io with a command as part of the web request.



192.168.0.11:5000/input?command=email

RPi Web Server - Check Email Time

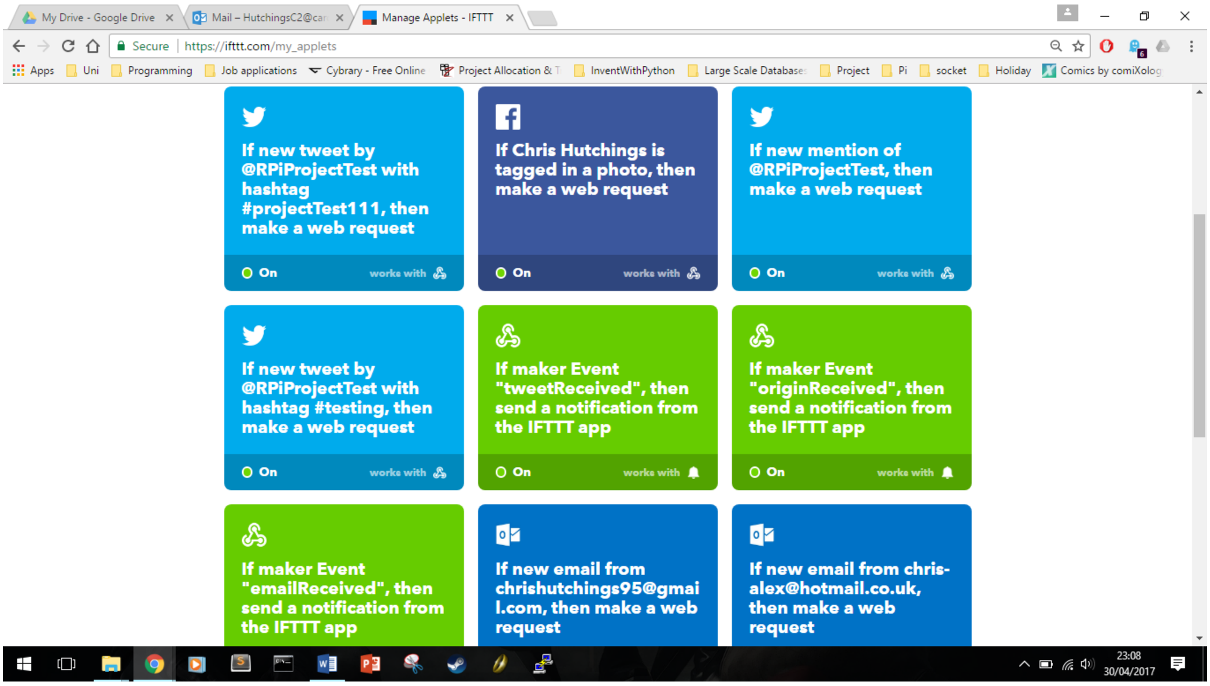
The date and time on the server is: 27-04-2017 13:44

Type `?command={command}` to listen for commands
e.g. 192.168.0.8/input?command=email

Go to a different page

- Led on/off
- Email Check
- Origin Check
- Tweet Time
- Tweet Origin

FIGURE 4.10 – FLASK WEB SERVER RUNNING ON THE RASPBERRY PI



If new tweet by @RPiProjectTest with hashtag #projectTest111, then make a web request

If Chris Hutchings is tagged in a photo, then make a web request

If new mention of @RPiProjectTest, then make a web request

If new tweet by @RPiProjectTest with hashtag #testing, then make a web request

If maker Event "tweetReceived", then send a notification from the IFTTT app

If maker Event "originReceived", then send a notification from the IFTTT app

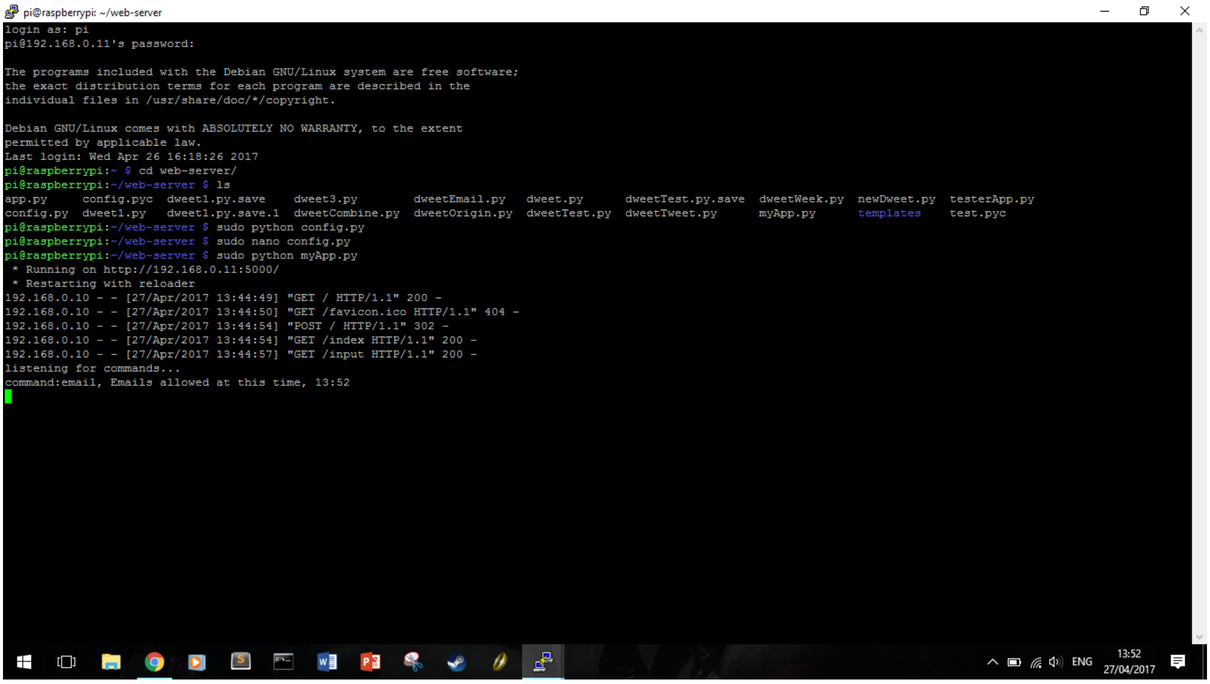
If maker Event "emailReceived", then send a notification from the IFTTT app

If new email from chrishutchings95@gmail.com, then make a web request

If new email from chris-alex@hotmail.co.uk, then make a web request

FIGURE 4.11 – APPLETS CREATED ON IFTTT AND USED FOR THE SOLUTION

The command sent was "email", the server was listening to Dweet.io with a real time subscription so when that command came through from IFTTT the server uses the command "email" to activate the policy in place. As presented in Figure 4.12, the command "email" was received at the time 13:52. According to the policy I established, emails are permitted between 9-5pm, therefore, this was allowed by the server. A green LED was lit up which can be seen in Gigure 4.13 and the notification applet on IFTTT was triggered, sending a notification to the user's phone. Evidence of this applet running can be seen in Figure 4.14.



```
login as: pi
pi@192.168.0.11's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Apr 26 16:18:26 2017
pi@raspberrypi:~ $ cd web-server/
pi@raspberrypi:~/web-server $ ls
app.py     config.pyc  dweet1.py.save    dweet3.py         dweetEmail.py   dweet.py      dweetTest.py.save  dweetWeek.py  newDweet.py  testerApp.py
config.py  dweet1.py   dweet1.py.save.1  dweetCombine.py  dweetOrigin.py  dweetTest.py  dweetTweet.py      myApp.py      templates    test.pyc
pi@raspberrypi:~/web-server $ sudo python config.py
pi@raspberrypi:~/web-server $ sudo nano config.py
pi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
192.168.0.10 - - [27/Apr/2017 13:44:49] "GET / HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 13:44:50] "GET /favicon.ico HTTP/1.1" 404 -
192.168.0.10 - - [27/Apr/2017 13:44:54] "POST / HTTP/1.1" 302 -
192.168.0.10 - - [27/Apr/2017 13:44:54] "GET /index HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 13:44:57] "GET /input HTTP/1.1" 200 -
listening for commands...
command:email, Emails allowed at this time, 13:52
```

FIGURE 4.12 – REMOTELY RUNNING WEB SERVER ON THE RASPBERRY PI THROUGH PUTTY

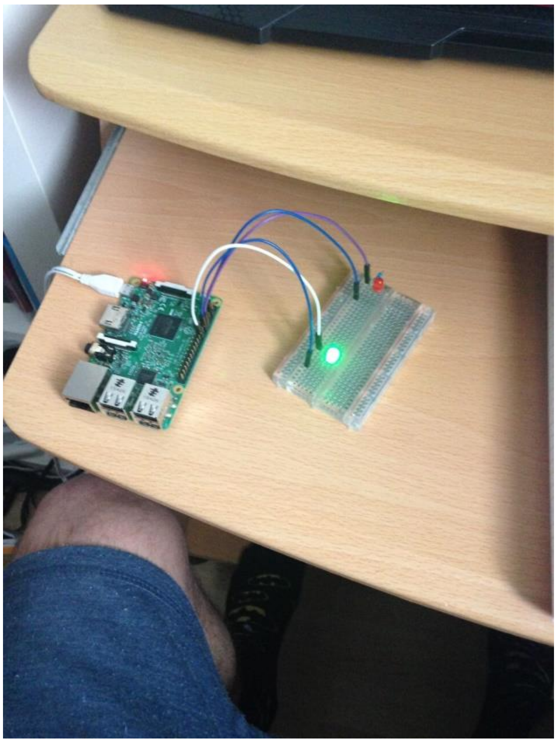
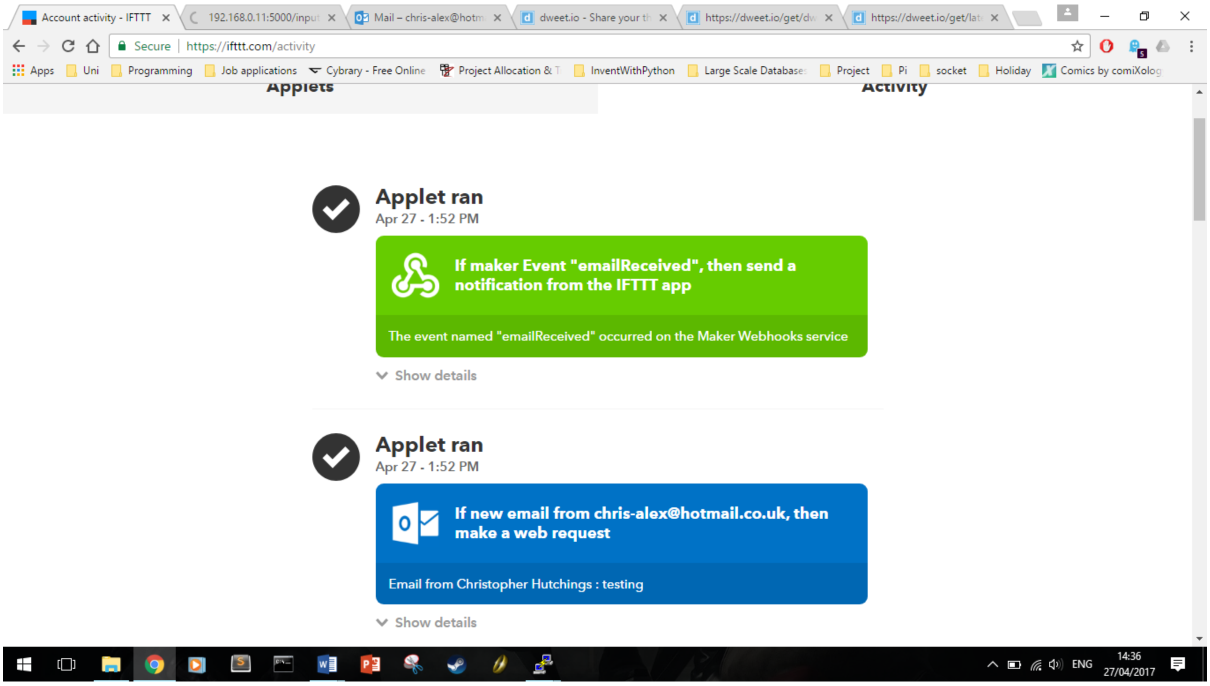


FIGURE 4.13 – GREEN LED TURNING ON WHEN POLICY IS ACCEPTED



Applet ran
Apr 27 - 1:52 PM

If maker Event "emailReceived", then send a notification from the IFTTT app

The event named "emailReceived" occurred on the Maker Webhooks service

Applet ran
Apr 27 - 1:52 PM

If new email from chris-alex@hotmail.co.uk, then make a web request

Email from Christopher Hutchings : testing

FIGURE 4.14 – NOTIFICATION APPEARING ON IFTTT SHOWING BOTH APPLETS RAN

Evidence of further testing of the policy in place can be found in section 5, Testing and in appendix 10.2.

4.8 Concept of the Solution

The entire concept behind this solution is that Twitter and Email are acting as a demonstration of what in fact could actually be implemented on top of other services on IFTTT, for example, Phillips Hue Lights, server room temperature and kitchen appliances. A Raspberry Pi is going to be used to give a visual demonstration through lighting up the LEDs attached to it in order to show what could be happening if a different IoT devices were used such as a set of Phillips Hue Lights. The solution being offered and the security layer implemented will provide protection to natural persons, data, integrity of office/home environments as well as the network implications, i.e. getting hacked and shadowing your personal details. The idea is that this solution could be implemented onto IFTTT's existing applet creation to help users create more secure applets by giving the users the ability to control and add policies to their applets to make them more secure. Applets are in existence for heating, network security and other day-to-day activities and this solution would provide credibility and protection for the expanding market.

To demonstrate the effectiveness of this solution and to take full advantage of IFTTT I will be using the Maker channel on IFTTT to send web requests through the application to Dweet.io in order to give the solution control over what functions and policies to execute and check against. For this concept to work I am using a combination of two IFTTT applets as shown in the above section, a trigger applet which triggers a web request to Dweet.io, and a second applet which is triggered after the policies in place are checked and successfully passed by the web server which make a web request then creating a notification on the user's phone informing them an event has occurred, for example, a new tweet or email.

Therefore this solution could be implemented onto IFTTT as an in-between layer for creating applets, allowing the user to select or create their own policies to add to each new applet they create. But for the case of demonstrating how the security layer works and is being implemented using IFTTT the solution will be using two applets for each different policy.

5 The Results

This section of the project discusses what the results of the solution implemented were, seeing how well the solution implemented actually does at solving the problem. It will also show visual results of how the system ran and what performance issues occurred, as well as the testing of the solution.

5.1 Testing of the Solution

In order to test that the solution I created was working as intended and did in fact meet the aims and objectives set in the beginning of the project I carried out tests. The tests consisted of two scenarios for each different applet created on IFTTT.

In order to test the time policy I have put in place, an email was sent from a Hotmail account to the linked email account on IFTTT, this triggered a web request to Dweet.io with the necessary command. The command was then sent onto the Flask web server which corresponds with the command sent to the server, executing the function which checks the time of the sent email is an allowed time period for emails to be sent and then if allowed, lights the green LED up and triggered the second applet on IFTTT notifying the user, else if it is not allowed then a red LED was lit up. This was tested with an email in the allowed time period and an email sent when the time period was not allowed.

For the origin of the source to be tested, the sender address must be in the trusted list in the configuration file and then that email address used to send an email to the linked IFTTT email address. If the address of the sender was not in the trusted list in the configuration file then it would result in the red LED lighting up on the Raspberry Pi and a message saying that the user was not a trusted source.

The applet that is triggered by Twitter was tested in much the same way as the email services. The first applet was triggered by a new tweet by the user @RPiProjectTest. If the time of the tweet being posted was not allowed within the time period specified by the user then the red LED would light up but if it was within the allowed time period then the green LED would be lit up and it would trigger a second applet to run on IFTTT notifying the user that an event had occurred.

The second Twitter applet was triggered a tweet with the hashtag #testing in it by user @RPiProjectTest. If the user was not in a trusted list then it would mean the Pi lights up red and if the user was in the trusted list then the Pi would light up green and trigger the second notification.

Test Number	Test Description	Expected Result	Actual Result	Pass or Fail
1	Sending an email from Hotmail account to IFTTT linked email at the time of 14:10	Green LED should light up with terminal message saying command, emails allowed at this time and a second applet triggering on IFTTT sending a notification to my phone	The Green LED on the Pi lit up for 10 seconds and the message “command: email, Emails allowed at this time 14:11 appeared. The second notification on IFTTT triggered notifying me	Pass
2	Sending an email from Hotmail account to IFTTT linked email at the time of 22:10	Red LED should light up with terminal message saying “emails not allowed at this time”	A message appeared saying: “emails not allowed at this time” and the red LED lit up	Pass

			for 10 seconds	
3	Sending an email from Gmail account to IFTTT linked email with the account chrishutchings95@gmail.com in trusted list in configuration file	Green LED should light up with terminal message saying that chrishutchings95@gmail.com is a trusted source and the second IFTTT applet triggering a notification on my phone	Green LED lit up on the Raspberry Pi for 10 seconds and a message in the terminal appeared, "command: origin The origin of the source is: chrishutchings95@gmail.com. The second applet then triggered sending my phone a notification	Pass
4	Sending an email from Gmail account to IFTTT linked email with the account chrishutchings95@gmail.com not in the trusted list in configuration file	Red LED should light up on the Raspberry Pi, with terminal message saying that "user is not a trusted source"	A message appeared in the terminal saying: "Origin of trigger is not a trusted source" and the red LED lit up on the Pi for 10 seconds	Pass
5	Sending a tweet on @RPiProjectTest with the text "Testing my project #testing" with RPiProjectTest in trusted list in configuration file	Green LED should light up on the Raspberry Pi for 10 seconds, with the message appearing in the terminal saying "command: Tweet Origin RPiProjectTest is a trusted source" and the second IFTTT applet triggering a notification on my phone	Message appeared in the terminal saying: "command tweetOrigin The user who triggered this event is: RPiProjectTest" and the green LED lit up on the Pi for 10 seconds, the second applet ran sending a notification to my phone	pass
6	Sending a tweet on @RPiProjectTest with the text "Testing the project again #testing" with RPiProjectTest not in the trusted list in configuration file	Red LED should light up on the Pi for 10 seconds with a message appearing in the terminal saying "source is not trusted"	A message appeared in the terminal saying: "user who triggered is not a trusted source" before the red LED lit up on the pi and stayed on for 10 seconds	Pass
7	Sending a tweet on @RPiProjectTest with the text "testing this pi" at the time 15:07	Green LED should light up on the Pi for 10 seconds and a message appearing in the terminal saying "tweets allowed at this time" and then the second IFTTT applet triggering with a notification appearing on my phone	Message appeared in the terminal saying: "command: tweetEmail, Tweets allowed at this time, 15:08. Green LED lit up for 10 seconds before the second applet was triggered sending a notification to appear	Pass

			on my phone	
8	Sending a tweet on @RPiProjectTest with the test "Testing my project again" at the time of 22:35	Red LED should light up for 10 seconds on the Raspberry Pi and a message appearing in the terminal saying "Tweets not allowed at this time."	A message appeared in the terminal saying: "Tweets not allowed at this time" and then red LED lit up for 10 seconds	Pass

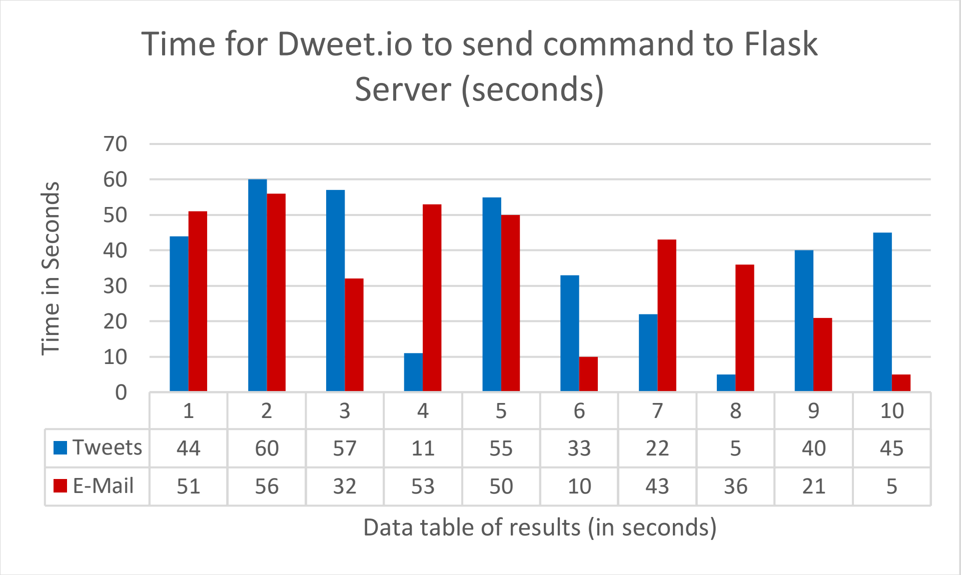
I decided on carrying out these tests to present how well the solution worked. This way I was able to demonstrate the whole solution was working together as intended, the trigger triggering the applet on IFTTT which would send the command through Dweet.io to the web server which activated the LED on the Pi and caused the second applet to trigger with the notification coming through.
There were two tests for each applet, a successful running of the full applet and a failed attempt of running the applet. This was because I had setup two scenarios that would happen, if the command was found and met the criteria set by the policy in place then it was successful or if the command did not meet the criteria set by the policy then it failed and showed the red LED.

To test the time period of both the email service applet and the Twitter service applet I used a list within the configuration file that consisted of the allowed times. The policy that I put in place was emails and tweets were allowed between 9-5pm (working hours). I chose this policy to demonstrate that a user could have time controls over when applets would be triggered and whether or not they were notified of this. For example if a user had a work email that was linked to IFTTT they may only wish to receive notifications of these emails to their phone during the work day preventing disturbances after the working day.

For testing the origin of the source, I added the Gmail email address to the trusted list in the configuration file and then sent an email from Gmail to the IFTTT linked email address. This triggered the applet to run and sent the command and the sender's email address through Dweet.io's service to the Flask web server. Once the server receives this command it then checked it against the policy in place for the origin command. As the email address was in the trusted list then it was allowed and the green LED lit up and triggered the second IFTTT applet to run notifying the user that an event had occurred. The origin of the source was a policy idea that I came up with to try and solve the issue that Dhanjani talked of in the background. This way the policy would stop any users that were not in your trusted list being able to attack your house or office light systems, for example as the applet would not be triggered due to the security layer and policies put in place.

5.2 Performance of Solution

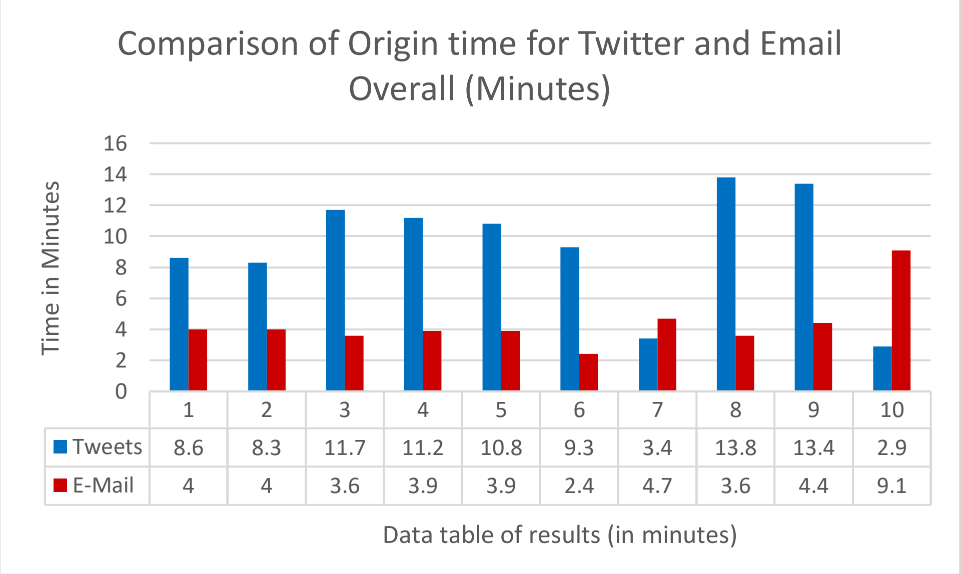
Throughout this project there were issues with IFTTT being slow to trigger the applets which made testing the solution quite difficult. In order to test IFTTT and the solution as a whole I created some bits of code to time the solution and the time it took for the command to be received by the web server. For this I timed how fast it took for the email applet for time to trigger and the solution to complete with a notification sent to the user's phone regarding the second applet being triggered. I did the same for the Twitter applet for time to complete the same task. I then repeated this for the origin applets of both email and Twitter.
The first performance test was the time it took for the server to listen and receive the command from Dweet.io in seconds. As you can see from the results they are quite similar in terms of time, a few results such as number 4, 8 and 10 are considerably different but I believe this is down to the services themselves (IFTTT and Dweet.io) where the request to the service was delayed.



	1	2	3	4	5	6	7	8	9	10
🟦 Tweets	44	60	57	11	55	33	22	5	40	45
🟥 E-Mail	51	56	32	53	50	10	43	36	21	5

Figure 5.1 – Graph of time comparing Origin times Twitter vs Email

The second performance test was testing the overall time it took for the whole solution to complete. In order to carry out this test, a timer was set when the function was executed, (in this case twitter origin and email origin) and a timer was started at the end of the completed solution and then end was taken away from the start time. After running this test 10 times the following graph was produced from the results seen in Figure 5.2. The results from the graph are surprising as when initially tested before timing the solution, Twitter was quicker as triggering applets on IFTTT and seemed to be communicating quicker than Gmail service. However, when observing the results of the graph I can see that Twitter clearly was slower. As seen from the results Twitter took well over 8 minutes for the majority of the performance tests whereas Gmail were mainly all under 4 minutes. IFTTT states that their applets have a 15 minute polling period and some of their services may take longer to update their API which can mean the applet is further delayed [18]. Therefore I believe that this is one of the reasons in why Twitter was taking so long to complete the task. The same can be said for the long email time of test 10, as it did get delayed and I had to send another email in order to trigger the applet and when I did do this, both applets triggered at the same time.

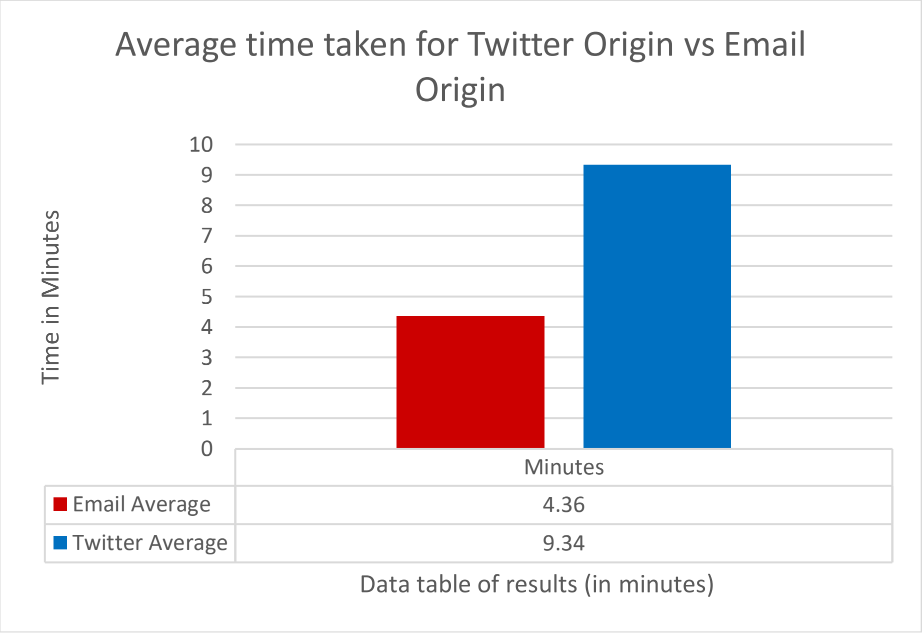


	1	2	3	4	5	6	7	8	9	10
🟦 Tweets	8.6	8.3	11.7	11.2	10.8	9.3	3.4	13.8	13.4	2.9
🟥 E-Mail	4	4	3.6	3.9	3.9	2.4	4.7	3.6	4.4	9.1

Figure 5.2 – Overall time it took for solution to complete Origin Policy

Looking at figure 5.3 the average time taken in minutes can be seen for both email and Twitter services. When tested the email applet triggered and sent the command to Dweet.io as well as the

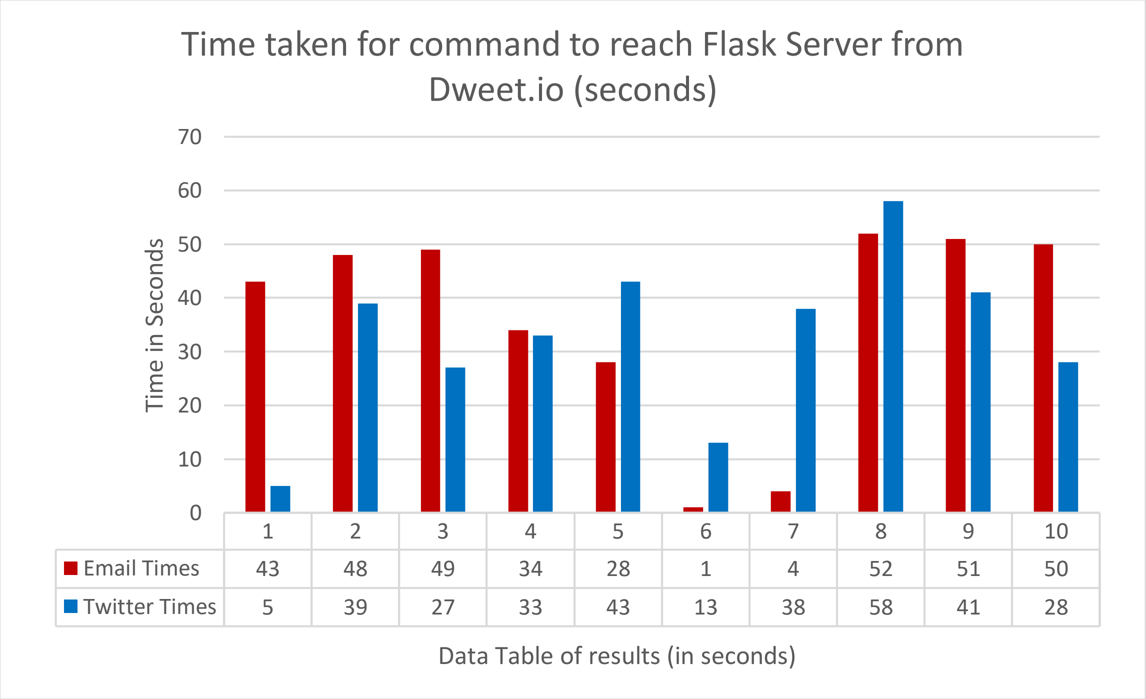
Flask server listening for that message on Dweet.io and controlling the Pi all happened just over 4 and a half minutes. Compared to Twitter which took just over 9 and a half minutes to complete the overall task on average. When testing the email service it worked flawlessly for each of the 10 test bar the last one which took 9 minutes making the average higher.



	Minutes
🟥 Email Average	4.36
🟦 Twitter Average	9.34

FIGURE 5.3 – AVERAGE TIME TAKEN FOR ORIGIN POLICY TO COMPLETE TWITTER VS EMAIL

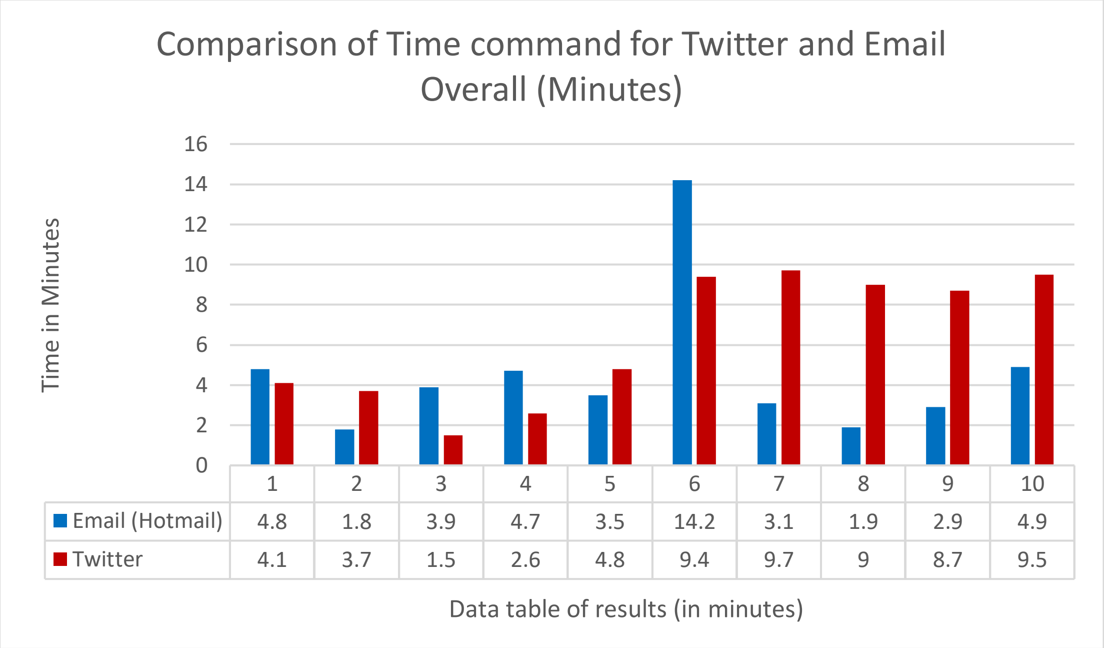
I then carried out the same performance tests on the time policy that I implemented. Testing which service was faster, Twitter or Email, when carrying out the overall task and which was quicker at receiving the command from Dweet.io. On average, Twitter was quicker at receiving the command from Dweet.io and sending it to the Flask web server but overall email was quicker. Similarly, with the above tests, the email (Hotmail account) applet triggered very quickly each time I tested it.



	1	2	3	4	5	6	7	8	9	10
🟥 Email Times	43	48	49	34	28	1	4	52	51	50
🟦 Twitter Times	5	39	27	33	43	13	38	58	41	28

FIGURE 5.4 – TIME IN SECONDS FOR FLASK TO RECEIVE COMMAND FROM DWEET.IO

Figure 5.4 shows that Twitter was quicker over half of the times in terms of listening and receiving the command from Dweet.io compared to the email applet. Whereas the overall solution Twitter was slower at completing the task. I can only speculate that the service was the problem at this point and Office 365 was working more quickly than Twitter was at the time in which I tested it. This could be that both of the services were facing delays on IFTTT, as the results are high due to polling by IFTTT to trigger the applets. This can be inferred as the time taken for Flask to receive the command after the applet was triggered was fast as seen in Figure 5.1 and Figure 5.4.



	1	2	3	4	5	6	7	8	9	10
🟦 Email (Hotmail)	4.8	1.8	3.9	4.7	3.5	14.2	3.1	1.9	2.9	4.9
🟥 Twitter	4.1	3.7	1.5	2.6	4.8	9.4	9.7	9	8.7	9.5

FIGURE 5.5 – TIME OVERALL COMPARISON BETWEEN TWITTER AND EMAIL FOR TIME COMMAND

Overall from timing the solution and receiving the results, on average the times waiting for Twitter to complete the solution for both tasks is 6.3 minutes for the time task and 9.3 for the origin task. Whereas, the email applet took 4.4 minutes to complete the origin task and 4.6 minutes for the time task, therefore, email was the quicker service when testing the solution. The reason behind timing the solution and discovering how quickly it performed was to find out which services and parts of the solution were causing delays to occur. As shown in the results the time taken to find the command on the web server is less than a minute, whereas the time taken for the overall solution is significantly higher. This can be seen as being down to IFTTT communicating with other services in order to trigger the applet and then sending the web request to Dweet.io before that is then received by the web server on the Raspberry Pi.

5.3 IFTTT Performance

Throughout this project I have mentioned that IFTTT had some performance issues. There is some speculation that IFTTT itself is quite a slow web service as it deals with lots of services and has a lot of application programming interfaces they must connect to and retrieve data from in order to check that a new email or a new tweet has occurred in order to trigger the applets. When an applet triggers there is an option to receive a notification when the applet triggers, on numerous occasions the notification was delayed by some time, at most up to 30 minutes later after posting a new tweet for example. Other times, the notifications came through in batches after posting two or three tweets together rather than after one. I believe this is because IFTTT polls the services after a certain amount of time (15 minutes according to a developer), this leads me to believe that if an applet is not triggered immediately it could well be because it has missed the polling phase and waits for the next polling phase to occur for that API. Again this is speculation but it seems to fit around what was occurring when using the service.

It also does say that requests are rate limited when creating the applet using a Maker service, therefore this could be the reason as to some delays when using IFTTT as the IP address has exceeded the rate limit within the time period and has to wait longer for it to occur.

5.4 Does the solution solve the problem

The solution currently takes two applets from the IFTTT application service and uses them both to simulate one applet but with an added layer of security in-between. The configuration file that is used and that is sat on the web server acts as a policy engine for the server allowing an applet to be triggered on IFTTT if the policies are met and not broken by the initial applet. A user (in this case myself) is able to add policies to the file which would be taken into account when using IFTTT.

If I look at the objectives set at the beginning of this report and compare the ones stated in the initial plan then I can see how well the whole solution actually solves the problem.

The objectives set at the beginning of this report are as follows:

- Create and observe applets on the IFTTT application – Objective achieved

The solution implemented uses email and Twitter services from IFTTT to demonstrate how security concerns can be overcome. How IFTTT usually works is that there is an applet which consists of a trigger and an action. After being triggered, an action is executed.

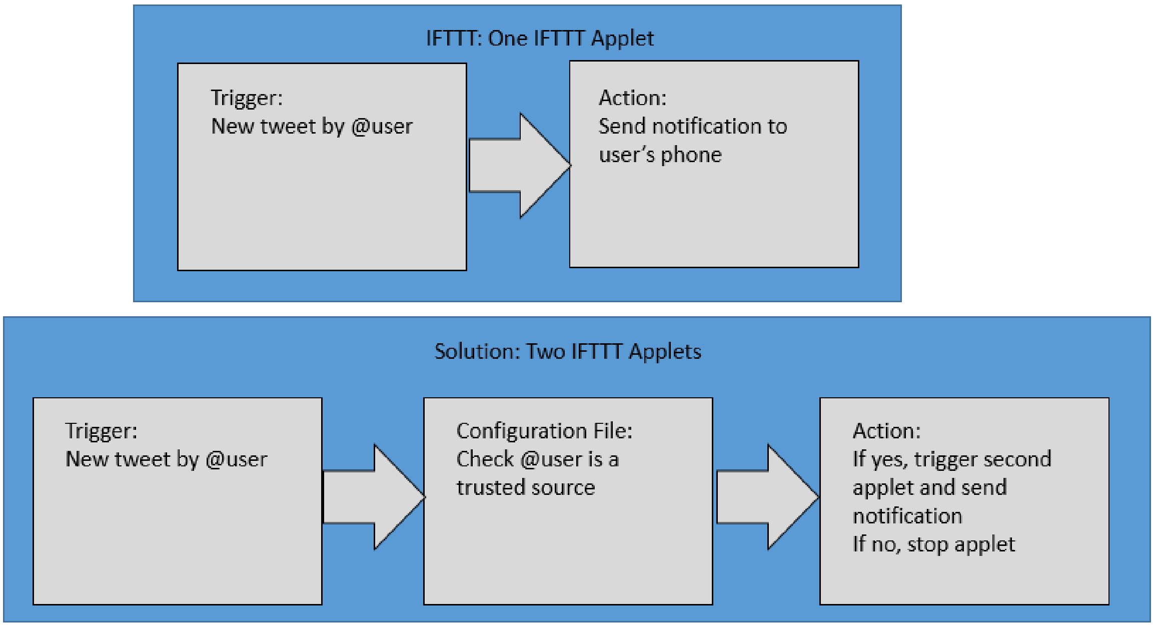

IFTTT: One IFTTT Applet
Trigger:
New tweet by @user
Action:
Send notification to user's phone
Solution: Two IFTTT Applets
Trigger:
New tweet by @user
Configuration File:
Check @user is a trusted source
Action:
If yes, trigger second applet and send notification
If no, stop applet

FIGURE 5.6 – SOLUTION COMPARED AGAINST CURRENT IFTTT

The solution implemented has added an extra applet in order to demonstrate how the use of a web server with a number of policies can be used to overcome security concerns previously highlighted. The solution provides assurance against such activity as notifications from unverified or untrusted sources and outside of specified time frames.

The evidence demonstrates that the solution has achieved the objective of using Twitter and Email services to demonstrate how specific security concerns can be overcome when using IFTTT, as the solution demonstrates that if you put in place the policies that you wish to have then the web server is able to control which applets notify the user and when.

- Learn how to setup a web server on a Raspberry Pi – Objective achieved

The second objective, learning how to setup a web server on the Raspberry Pi, using Flask micro framework in order to remotely control the GPIO pins was achieved. This is clearly demonstrated by the successful installation of the web server onto the Raspberry Pi. When a command is received by the Pi LEDs on the Pi light up as shown in the diagram below. The Pi responds to the policy and the specific criteria must be met to allow the continuation of the process.

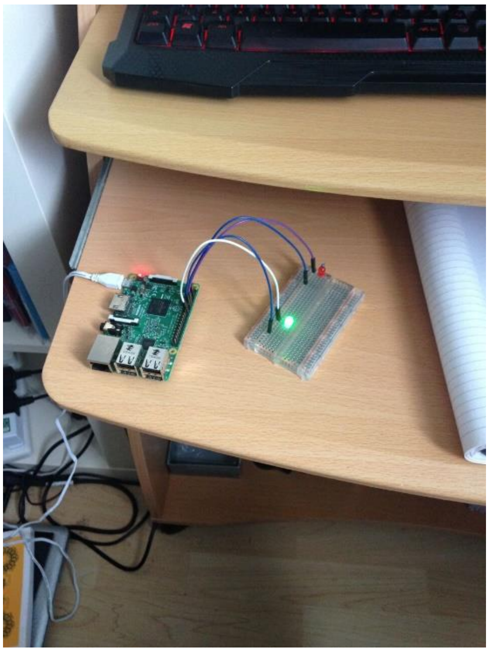
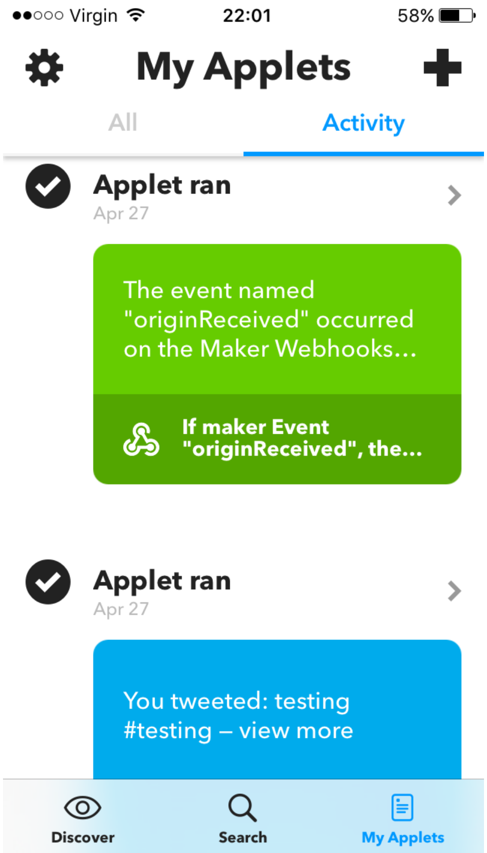


FIGURE 5.7 – RASPBERRY PI LIGHTING UP THE GREEN LED AFTER SUCCESSFUL POLICY



Applet ran
Apr 27

The event named "originReceived" occurred on the Maker Webhooks...

If maker Event "originReceived", the...

Applet ran
Apr 27

You tweeted: testing #testing – view more

FIGURE 5.8 – NOTIFICATION APPEARING ON MOBILE AFTER SUCCESSFUL POLICY EXECUTION

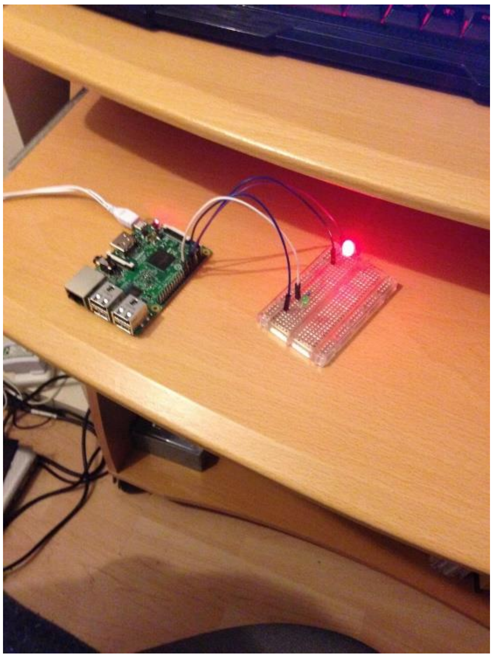
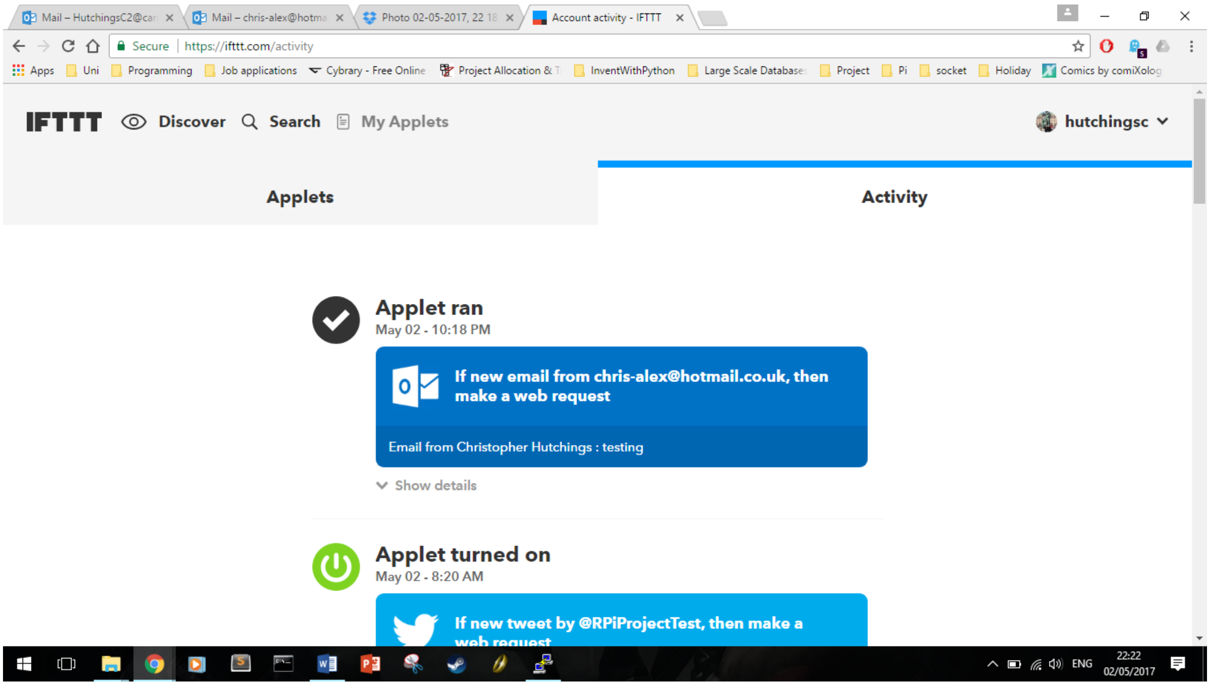


FIGURE 5.9 – RED LED TURNING ON AS POLICY CRITERIA WAS NOT MET



Applet ran
May 02 - 10:18 PM

If new email from chris-alex@hotmail.co.uk, then make a web request

Email from Christopher Hutchings : testing

Applet turned on
May 02 - 8:20 AM

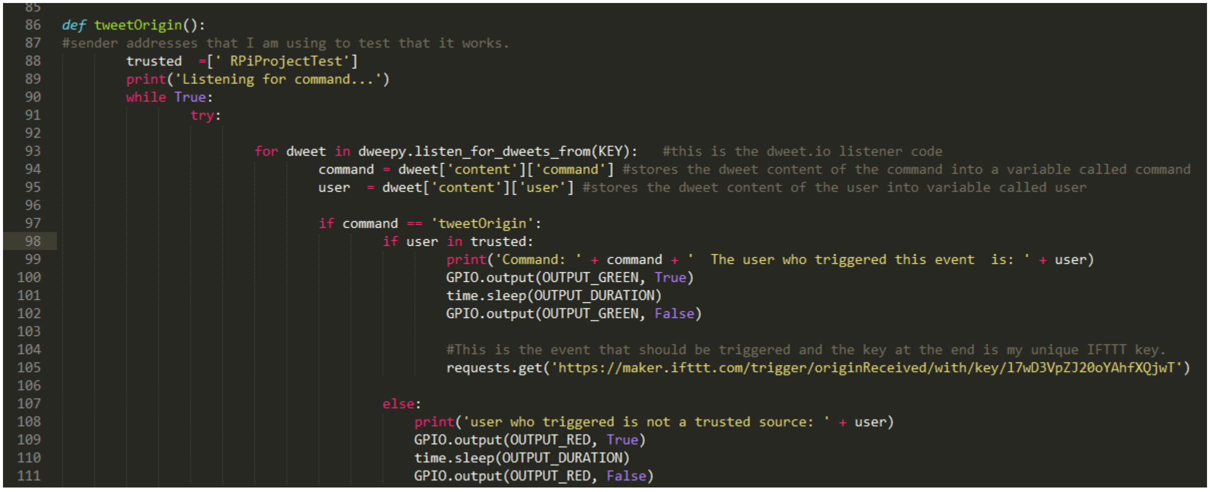
If new tweet by @RPiProjectTest, then make a web request

FIGURE 5.10 – APPLET RUNNING, BUT NOT GOING FURTHER AS THE POLICY CRITERIA WAS NOT MET

As demonstrated in the above Figures, when the applet is triggered and meets the criteria set for the policy, a green LED is lit up and the notification applet is triggered, sending the notification to my phone. This shows the interaction between the Web server and the Raspberry Pi. Furthermore, when the applet is triggered and does not meet the criteria set by the policy, the red LED lights up with no follow on applet being triggered and no notification being received by the user. Therefore, this demonstrates how the objective of getting the web server set up and interacting with the Raspberry Pi has been achieved.

- Explore security policies that could be implemented - Scope altered as discussed in 4.1 all other deliverables achieved

The redefined third objective has been achieved, evidence through successful demonstration of security policy in operation. The security policies tested i.e. origin and time have been implemented in this solution and shown to be working well. The origin of the source is verified by using a list within the configuration file for the user to add trusted sources to and if the user is not in this list then the source is not trusted or verified by the server. This, therefore, is successful evidence of completion of this aspect of the objective. The time periods that are allowed by the policies in place on the server work also. By using a list of allowed times which are checked when the received command arrives to the server, if the time the email was sent or the tweet posted are within one of those hours allowed in the list then the server will accept the action and trigger the second applet, otherwise the response is a red LED and no further action.



```
def tweetOrigin():
#sender addresses that I am using to test that it works.
        trusted  =[' RPiProjectTest']
        print('Listening for command...')
        while True:
                try:

                        for dweet in dweepy.listen_for_dweets_from(KEY):   #this is the dweet.io listener code
                                command = dweet['content']['command'] #stores the dweet content of the command into a variable called command
                                user  = dweet['content']['user'] #stores the dweet content of the user into variable called user

                                if command == 'tweetOrigin':
                                        if user in trusted:
                                                print('Command: ' + command + '  The user who triggered this event  is: ' + user)
                                                GPIO.output(OUTPUT_GREEN, True)
                                                time.sleep(OUTPUT_DURATION)
                                                GPIO.output(OUTPUT_GREEN, False)

                                                #This is the event that should be triggered and the key at the end is my unique IFTTT key.
                                                requests.get('https://maker.ifttt.com/trigger/originReceived/with/key/l7wD3VpZJ20oYAhfXQjwT')

                                        else:
                                            print('user who triggered is not a trusted source: ' + user)
                                            GPIO.output(OUTPUT_RED, True)
                                            time.sleep(OUTPUT_DURATION)
                                            GPIO.output(OUTPUT_RED, False)
```

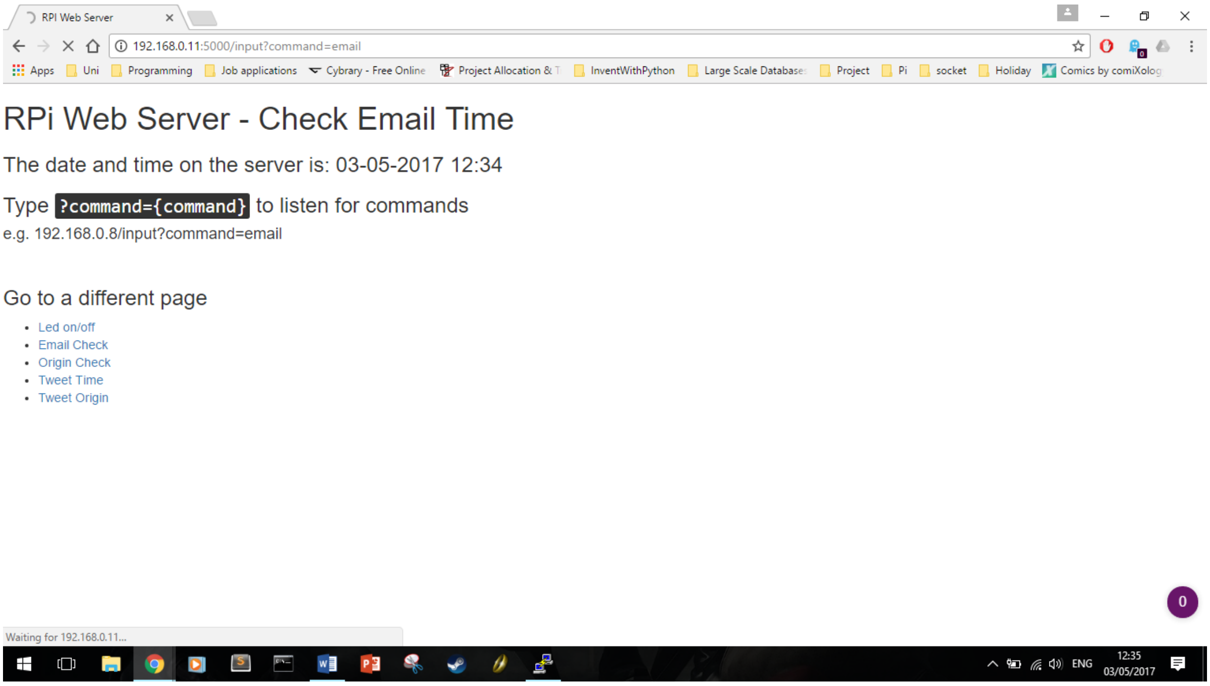
FIGURE 5.11 – SNIPPET OF POLICY FOR VERIFYING A SOURCE

The last part of this objective was to look into encrypted channels. The idea behind this was to allow a user to encrypt an event so that only they knew what the event was until a certain time or day. For example I may encrypt an event so that I receive a notification saying an event has occurred but if I was to look at the event I would not know what it was. This was seen to be an idea that could be used for groups of people, showing a certain group the event during the week but only certain other users could see the event that was generated on the weekend. As aforementioned, this could be achieved by implementing user groups which would allow the admin of that group to choose which users can view events. This is the next stage of the project which can be carried out in future work.

- Demonstrate how interaction with IFTTT, a Raspberry Pi and a web server can be used to simulate what could be done with other Internet of things enabled devices – Objective achieved

Fourthly, the last objective that was set at the beginning of this project was to demonstrate how this solution could be used with other devices on the IoT such as the Phillips Hue Lights. This objective has been met. The solution could be used with other IoT devices.

The overall solution would suit other IoT enabled devices and be easily transferrable in order to test that the solution with the added layer of security would in fact work with these other devices. Further testing would be required in order to demonstrate how the solution would work with the other IoT enabled devices which can be carried out in future work. Below are figures that provide evidence of the solution working with a different IoT enabled device.



RPi Web Server - Check Email Time

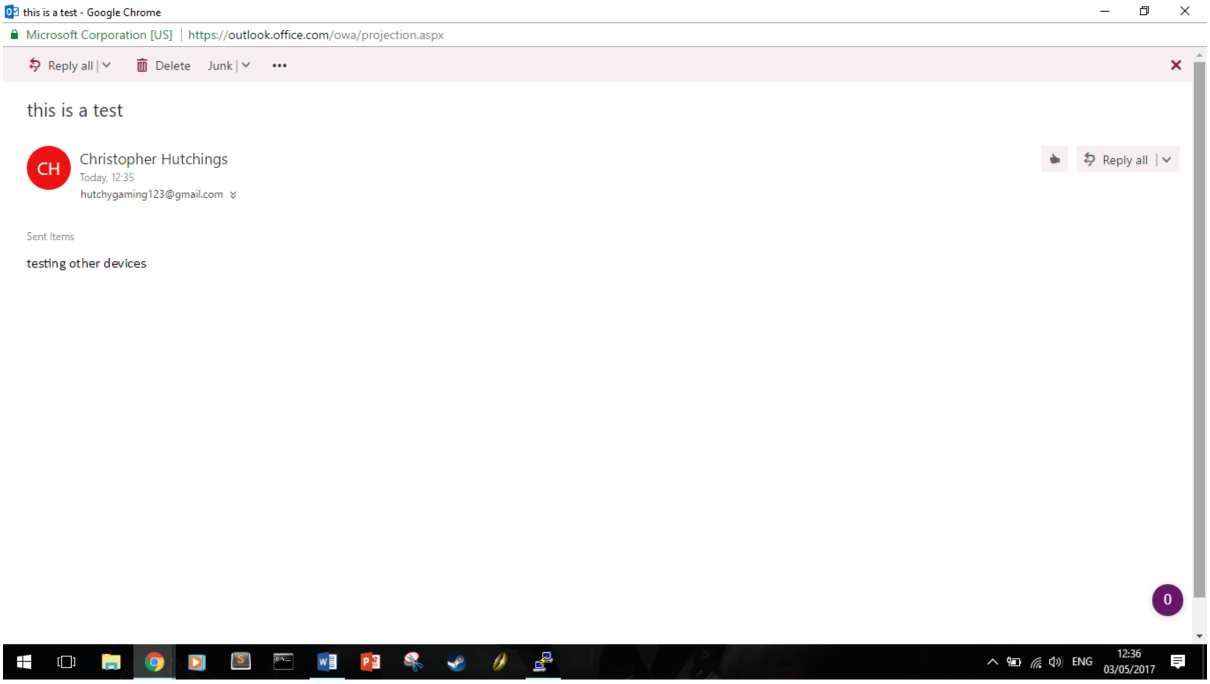
The date and time on the server is: 03-05-2017 12:34

Type `?command={command}` to listen for commands
e.g. 192.168.0.8/input?command=email

Go to a different page

- Led on/off
- Email Check
- Origin Check
- Tweet Time
- Tweet Origin

FIGURE 5.12 – SERVER RUNNING LISTENING FOR THE EMAIL COMMAND


this is a test - Google Chrome
Microsoft Corporation [US]
https://outlook.office.com/owa/projection.aspx
Reply all
Delete
Junk
Reply all
ENG
12:36
03/05/2017

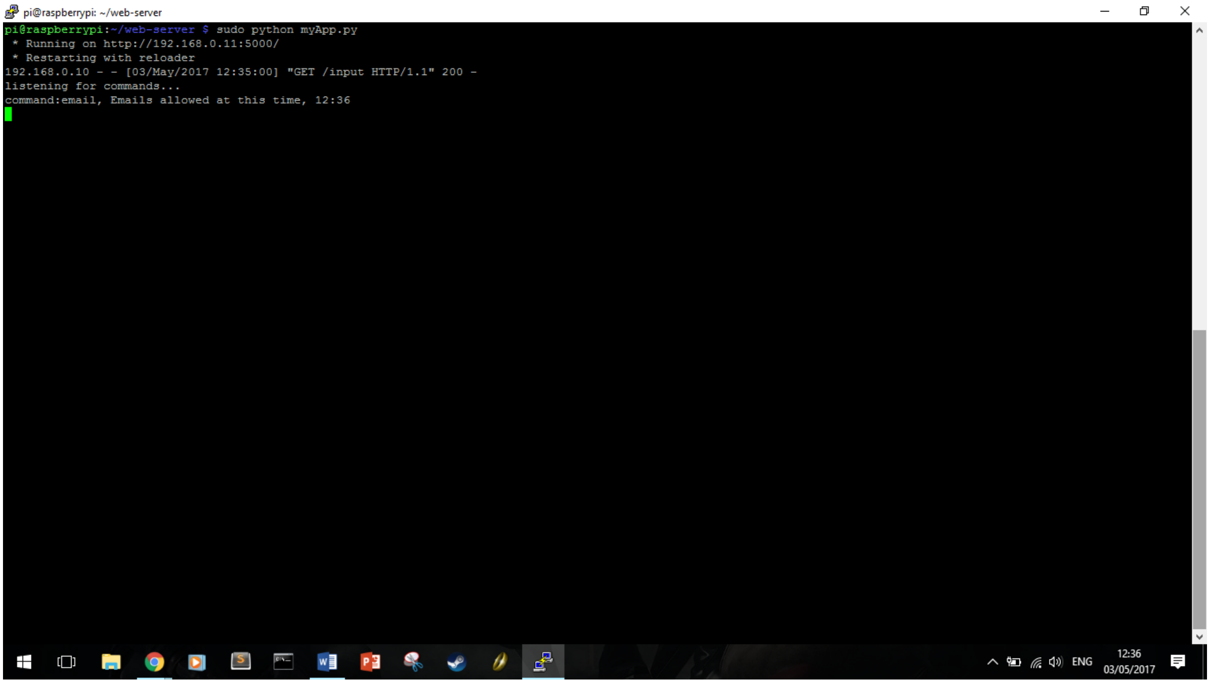
this is a test

Christopher Hutchings
Today, 12:35
hutchygaming123@gmail.com

Sent Items

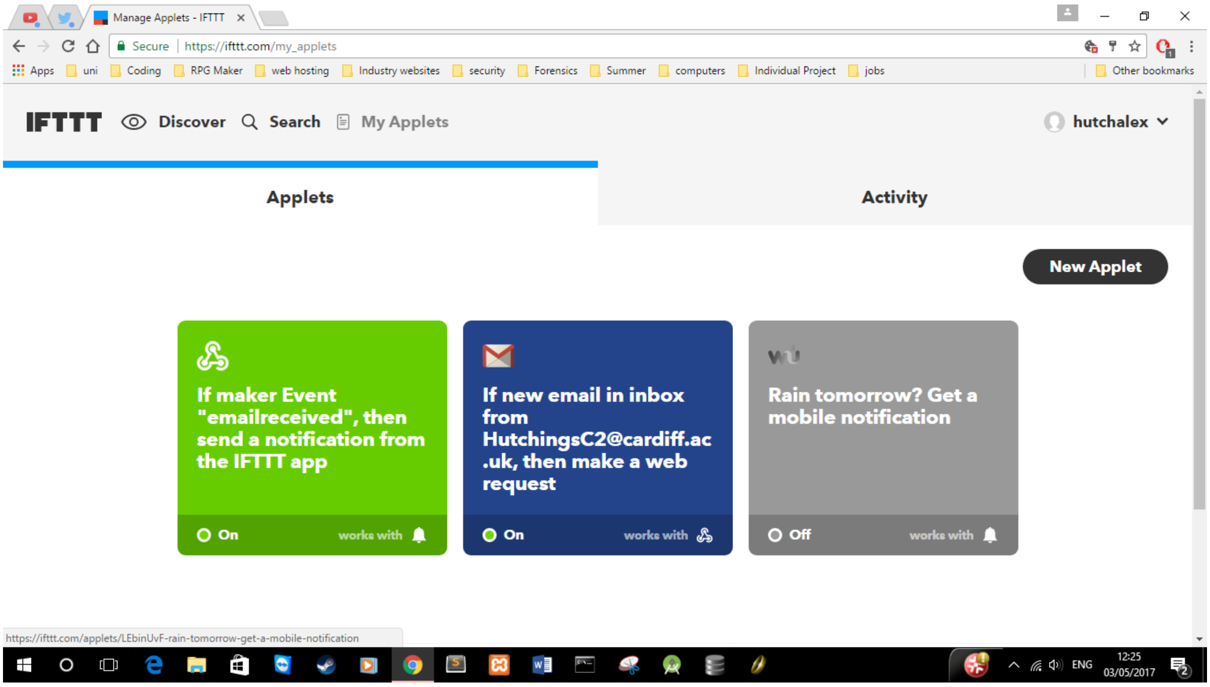
testing other devices

FIGURE 5.13 – AN EMAIL BEING SENT TO A DIFFERENT IFTTT ACCOUNT EMAIL ADDRESS



```
pi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
192.168.0.10 - - [03/May/2017 12:35:00] "GET /input HTTP/1.1" 200 -
listening for commands...
command:email, Emails allowed at this time, 12:36
```

FIGURE 5.14 – THE COMMAND RECEIVED AND IT PASSES POLICY CRITERIA



If maker Event "emailreceived", then send a notification from the IFTTT app
○ On works with

If new email in inbox from HutchingsC2@cardiff.ac.uk, then make a web request
○ On works with

Rain tomorrow? Get a mobile notification
○ Off works with

FIGURE 5.15 – A DIFFERENT IFTTT ACCOUNT USING THE SOLUTION

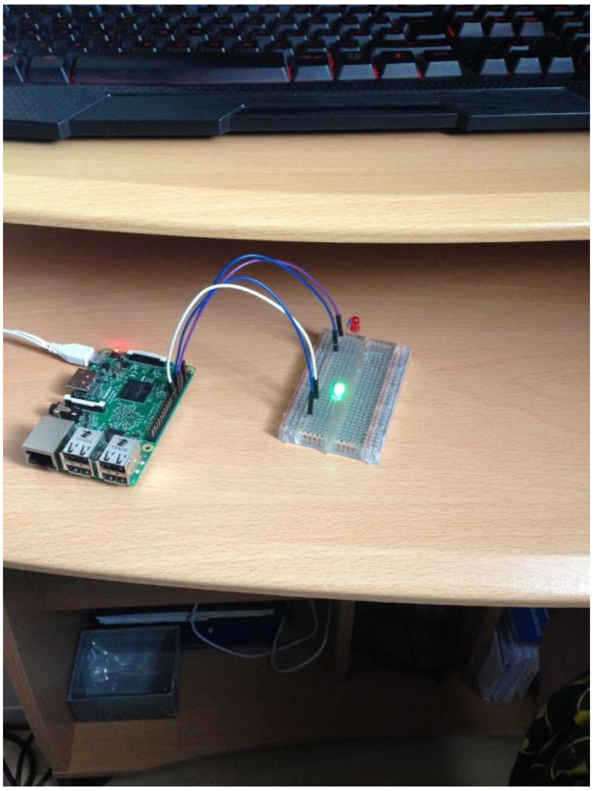
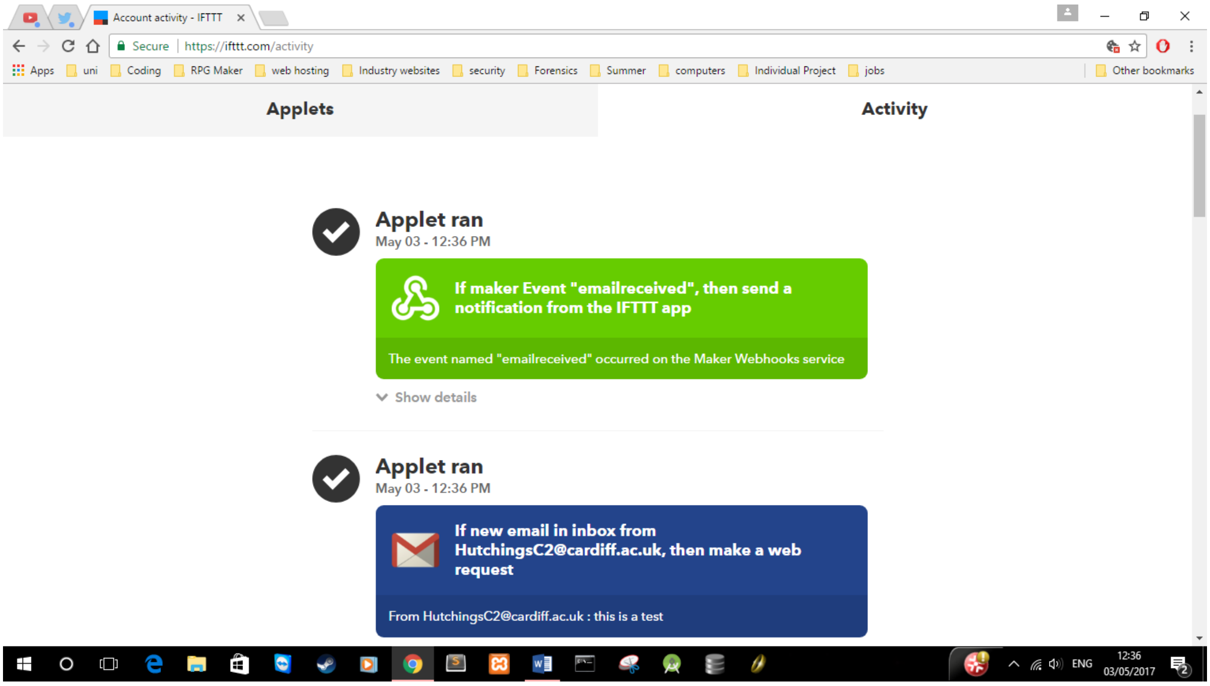


FIGURE 5.16 – GREEN LED LIGHTING UP AFTER SUCCESSFULLY RUNNING ON DIFFERENT DEVICE



Applet ran
May 03 - 12:36 PM

If maker Event "emailreceived", then send a notification from the IFTTT app

The event named "emailreceived" occurred on the Maker Webhooks service

Show details

Applet ran
May 03 - 12:36 PM

If new email in inbox from HutchingsC2@cardiff.ac.uk, then make a web request

From HutchingsC2@cardiff.ac.uk : this is a test

FIGURE 5.17 – THE SOLUTION RUNNING AND WORKING ON A DIFFERENT IOT DEVICE

The Figures 5.12-5.17, show the solution running and successfully working on a different IoT enabled device. To demonstrate that the solution was easily transferrable to other IoT enabled devices I used a different mobile phone and a different IFTTT account. On the new IFTTT account I had the user create two applets, as seen in Figure 5.15. The web server was then run, listening for the email command to come through. By sending an email to hutchygaming123@gmail.com from Hutchingsc2@cardiff.ac.uk, the applet on the account was triggered and as it met the criteria of the policy in place the green LED was lit up, and the second notification applet was triggered sending a notification to the user’s mobile phone. Therefore, this demonstrates clearly that the solution has been used successfully on a different IoT enabled device and thus the objective has been achieved.

Overall, the solution solves the problem identified at the outset. The solution provides a bridge to the gap between IFTTT and IoT devices. The solution demonstrates how using a web server with a configuration file and a Raspberry Pi can visually show that implemented policies can control applets on the IFTTT application. By using the tools provided to intercept malicious types of triggers by attackers on unsuspecting users, the solution provides an effective layer of added security and will provide assurance for users in the ever-growing and developing market.

6 Future Work

This section will be discussing how the project would be carried on and the next steps needed in order to continue development and work on this project to further strengthen the solution. It will also discuss the different directions that I could have gone into or things that I could have done differently throughout the project.

6.1 Next Steps

Currently, the solution implemented is only a brief introduction into this topic and what could be achieved with this kind of solution. The concept has been put into practice and opens the way for more ideas like this to be taken further. The security layer itself can be adapted for individual automation applications and it could also be taken in a more complex route using Twitter APIs and other APIs to allow for more control of the solution, for example rather than having a list of trusted sources that list could be your Facebook friends list or your follows list on Twitter.

The next step I would take in further developing this project would be to create a more user based solution by adding a database which would allow a user to create an account and therefore allow a user to log in and create their own customised policies and rules. This would also tie in with another future development of creating a website with a web interface that allows users to create their own policies and rules on the website, rather than having to edit the configuration file. Each user would then be able to make their own personal policies for IFTTT to follow in order to trigger notifications from the application.

Another future step I would like to implement would to be test out the solution created with a number of different domestic devices for the IoT as this seems to have the biggest potential because of the growing market. These devices would include Phillips Hue Lights and other home based devices that could be used to test the security of IFTTT when using these devices, such as microwaves and ovens. The solution would need to be configured for each specific device in order to make sure it works correctly as this solution is configured for use with a Raspberry Pi and a web server on the Raspberry Pi itself. Therefore, in order for the Phillips Hue Lights to work I would need to connect to the Phillips Hue website and get a private API key to allow control of the lights whereas with the Raspberry Pi I did not need a private key.

Furthermore, one could look into the creation of different policies that could be implemented to help make the security layer more varied. As discussed in the implementation section, encrypted channels would be an interesting way of making different users and different groups of users being able to see certain information on specific days depending on the group that these users were in. This would mean giving users the ability to group other users on IFTTT. Similarly, this could work by incorporating policies that Microsoft use in their automation application, to allow only certain services access to data at different times or restrict access to services based on location, type of data and other policies that intend to keep the data private from specific groups of users or services. This would definitely improve the solution by allowing more restrictions and options for policies to give to a user to allow for more control over what they want their applet to do.

Currently the solution is only intended to work with IFTTT. A future implementation could be to get it to work for any automation application that is on the market. To do this there would be a general blanket solution that would then be customised depending on the service that the user chose to use. The configuration file would be full of different functions for each different service and then depending on the one chosen, it would use the functions related to that service, for example if you chose to use Stringify applets then the web server would direct the triggers and events to the Stringify section of the configuration file.

6.2 Different Directions

A different direction that this project could have gone in would have been to use a different Internet of Things device such as a Phillips Hue Light and customise the security layer around this. Using this device instead would have allowed me to overcome specific threats presented with the lights as mentioned in the beginning of the report. As well as Phillips Hue I could use WeMo home automation as that consists of a lot of different IoT enabled devices which can join your home network and be controlled using IFTTT. This would be a good way to compare both Hue and WeMo and test different scenarios in which the security layer can prevent attacks on these types of devices used within the safety of our homes. Using these devices would mean that different services on IFTTT could be used rather than Twitter and Email as demonstrated with this project. Both Hue and WeMo have their own services on IFTTT and could be used to help control the devices connected to them.

The whole project focuses solely on making a security layer for If-This-Then-That. After using IFTTT for the entire duration of this project and encountering issues in terms of performance speed and delays, as well as using Dweet.io as an intermediary to receive the command from IFTTT to the web server. I think a direction I could have gone in would be to remove IFTTT and Dweet.io and make a solution for the individual services instead. By creating a specific solution for each service such as Twitter, this would work by using Twitter's API, when posting a tweet or receiving a tweet it could run it against a web server policy much like the current solution does but instead without IFTTT and Dweet. It would see if the tweet meets the policies put in place by the user. This direction focuses more on social media and implementing policies and rules to control those platforms rather than IoT enabled devices but it would be an interesting way to incorporate some of the techniques used in this project and see how they can be used to monitor and control social media.

6.3 Solution Improvements

The solution in its current state works but could be further improved upon. One of the main improvements I believe that could be implemented would be to make the web server interaction more user friendly and easier to use. The web server needs the user to input a command into the URL bar in order to start listening for the command sent by Dweet.io. In order to meet the objectives set out in this project I had to learn how to use the web server from the beginning. Therefore the solution is not as complex or user friendly as it could be with further resources. I think it would be better if the user did not have to manually enter the command into the URL bar but clicked a button that would effectively automating the task. I think this would improve usability of the system immensely and would be more efficient as well because if the user misspells the command in the URL bar then they are directed to a page where it redirects the user to an erroneous page. Therefore this would be an improvement I would make in the future with further resources. I would like to continue to learn about web servers to give greater control and further design ideas.

Another improvement I would like to make to the solution would be to try out different orders in terms of IFTTT applets. Rather than having an applet triggered by a tweet first I could have a Maker web request first which posts a tweet to Twitter which would then trigger a second applet which makes a web request to Dweet.io which sends the command to the web server and see if this increases or decreases the time delay with IFTTT. The logic behind this is that rather than having IFTTT poll the Twitter service, the Raspberry Pi would make a get request to IFTTT triggering that applet instead.

Alternatively, another avenue to explore would be to remove Dweet.io from the processes. This I believe would decrease the time it takes for IFTTT to send the command to the web server itself. In order to carry out this improvement the Raspberry Pi Flask web server would be the publicly accessible URL address rather than Dweet.io and the server would access the command sent as part of the IFTTT web request in the same way it does with Dweet.io now but the command would be accessible straight away from the web server rather than having to go through another service.

7 Reflection on Learning

This section discusses how well I have done on this project, whether or not I went in the wrong direction and changes I made, as well as some reflections on the project itself and reflecting on the choices I made.

7.1 Project Evaluation

The project itself has been a success I believe with the solution implemented meeting the majority of the objectives set. The aim of the project was to create a web server which acted as a policy engine creating a security layer for the IFTTT application. Therefore, overall I would say the project has been a strong success as it does meet the aim set at the beginning of the project and it does meet the objectives of using two of three services on IFTTT, with regards to encryption of channels being looked at in future work. It also could be used as part of a solution for other devices although they have not been tested with the current solution. In order for the project to be a more rounded success, these factors could also have been explored. On reflection it would have been interesting to research further domestic devices that could be used in this field.

There have been times throughout the project where I went in the wrong direction and needed to adapt in order to ensure that the project was a success. Mainly this happened when using Apache as the web server towards the beginning of the project. As mentioned throughout this report and in the self-evaluation, I struggled with Apache and I believe if I had not changed direction and used Flask instead then the project and solution implemented would be very different to what it is now. An example of the difficulties encountered when using Apache came when I created a Python script which used sockets in order to communicate between a client and a host, the host being setup on the Raspberry Pi, a laptop acting as a client and sending commands this way. This was not the image I had thought of or decided upon when designing the solution and thus led me to change direction as I felt I was losing sight of my original objectives.

In its current guise, the project relies on too many services that are not under its control. It uses IFTTT, Twitter, Gmail, Hotmail and Dweet.io, all of these services are necessary in order for the project to act as a security layer between applets run on IFTTT. However, Dweet.io could have been removed if I had focused on making the Raspberry Pi Flask server available publicly towards the beginning of the project rather than towards the end, as this would have removed the necessity of having the Dweet.io service to relay the command from IFTTT to Flask. At times when testing the solution, waiting for a response from IFTTT did take time and not knowing whether or not that was down to IFTTT or a different service such as Gmail could cause frustration for end users. The code for the solution was correct and working it just took time due to IFTTT applets being polled and the possibility of them being delayed.

In order to overcome some of the problems encountered, I carried out extensive debugging throughout the project. There were times when the code ran smoothly and times where the code was slow in response. In order to debug the solution, I tested each individual policy without the use of the server. This consisted of setting up a channel on Dweet.io and an applet on IFTTT with a condition in place within a Python file to see if it worked as expected or if the problem was recreated. Once I was happy that the problem was fixed in that section of code, I added it and tested it using the server. I carried this out for each individual part before they were all integrated with the server.

IFTTT does not provide error messages which made debugging this solution rather difficult as if IFTTT was not working then there was no message received notifying me of this. Therefore, this made it difficult as sometimes I was just waiting for IFTTT to trigger the applet that I knew had worked previously. The only inclination I had of IFTTT not working was that the applet was not triggered within a 15 minute period then I knew that something was wrong, whether it was the internet connection I was using at that point in time, the request load on IFTTT or whether the network I was using was unable to send requests to IFTTT because too many had been sent, limiting me. It should be noted that these instances were low in frequency.

Debugging the solution through the Raspberry Pi itself was a challenge. This was because I was using putty to edit and manage code for the server file and the configuration file. This made it difficult as I was writing code on a Windows machine and then transferring it across to a Debian operating system on the Pi, which made errors in the code mostly to do with indentations. The Raspberry Pi itself was easy to setup and debug, getting the GPIO control was straight forward after seeing some tutorials on how to do it. The Pi was also an indication to see if the solution was working, because if the solution was working as intended then one of the LEDs attached to the Pi would light up therefore providing an indication whether or not the solution had completed its task. The main challenge with debugging the Raspberry Pi was the use of the nano editor. It did not show or highlight parts of the code that were wrong, such as variables that were not declared or spelt wrong until I ran the code. It also does not give line numbers on the side of the editor which did make finding the error time consuming when a lot of the code for this solution is quite similar.

7.2 Self-Evaluation

Overall I am pleased with outcome of this project as I have created a solution that has potential to go on and be more useful to other students and lecturers investigating and communicating with devices that are enabled for use within The Internet of Things. I managed my time effectively throughout the project. I set myself weekly targets with deliverables and ensured I tracked my progress each week for the duration of the project. With a larger time window I would like to investigate the potential of this solution in order for it to become more finely tuned and user friendly. Ultimately, I have achieved what I set out to do at the start of the project.

At the start of the project I had never used a Raspberry Pi before, I had not created and run a web server before and I had not used Python to send get requests from websites or communicate with a web server before. After this project I can safely say that I have now done all of these things. I realised the Raspberry Pi could be used as a web server to act as the gateway between IFTTT and Raspberry Pi controller itself to demonstrate how the security layer would work in practice. At first, getting the Raspberry Pi to work was fairly simple and easy and setting it up as a web server was not too complicated after following a number of tutorials online, however then the process began to become more complicated due to the aforementioned difficulties with Apache. Again, this was a new process and getting the file permissions themselves on the Raspberry Pi, making sure each file had the right group settings and adding them to Apache's configuration file proved to be difficult however, this led me to use the more user friendly Flask. Remotely controlling the GPIO pins for the Raspberry Pi was simple with Flask and after a couple of online tutorials and forums I had managed to get the code setup to handle this, which helped me to learn what is needed to control the pins on the Raspberry Pi and making sure to set the mode and the output for the correct pins on the controller itself.

Setting up the web server was a new learning process. Using Flask for the majority of the project helped in learning how to communicate with a web server using Python and a Raspberry Pi. Giving each page of the application a path and name so that the web server could correctly find the pages as well as creating the templates in HTML which were used for the design of the pages was an interesting new experience and getting these to work together was tough and took some time in order to understand what was happening in each part and how the Flask server was calling on the templates folder to generate the same layout for each page (if you used the same template).

Integrating the server with IFTTT and Dweet.io was probably the hardest part of this project. In order for the IFTTT applet to trigger a Maker web request which was what I needed in order to add my own policies and rules as it gave me control of what happened with the data sent from IFTTT. This took a lot of time researching until discovering the solution of using Dweet.io. During further investigation I discovered that I could have made the Raspberry Pi publicly available, however, as I only discovered this towards the end of the project I decided to keep Dweet.io in the solution as it would have changed too much at a late stage of the solution development. With further resources I could further the work already completed and improve the solution that I have developed. I believe this is a genuine gap in the market and would like to continue my research into the field.

Throughout the project I think I have worked well and kept on track, having to make changes from the initial plan of the project as the concept and idea of what I actually wanted to implement fell into place more clearly. At the beginning of the project, the end result had not fully crystallised but after some deliberation the concept and idea of what I wanted to achieve with this project and the solution I wanted to create became clear, even though some of the tools I used did change, the idea was still the same and the goal was still the same it was more a matter of finding the correct tool for achieving the goal, which eventually I did.

8 Conclusion

In conclusion, this project has achieved the first step in the creation of the original concept of adding a security layer to the application If-This-Then-That. I think that overall the solution that has been implemented works well and can be used to add policies and rules to a configuration file which gives the user more control over applets. This way, the applets on IFTTT must then abide by the policies and rules set up by the user in order to get the notification through to the user on their device.
The project itself does achieve the aim and the majority of the objectives set at the beginning of this report. Therefore, I would consider the project to be a success and a genuine solution to the problem of cyber security within the market of increasing demand concerning IoT.

From the results there is some evidence of problems and delays when using IFTTT and the other services. As mentioned in the future work, I would like to continue this project by refining these problems and reducing the number of services that the solution requires to run and complete its tasks. I believe that this would help improve the overall running of the solution and the way the solution works entirely. The current solution that has been implemented is a success in terms of what I set out to achieve at the beginning of this project but as stated it could have been improved and focused more on usability and making it work in a different way. However, for its intended purpose and for this project I believe that it works well. The solution can be used to help solve the problem of security issues when using automation applications, in this case If-This-Then-That and the Internet of Things.

In order to fully comprehend the success of this project you must look at how the solution has the potential to work with other devices that are IoT enabled. The possibility of adding a security layer and the idea of using a policy engine like the one implemented with other devices would help make them a lot more secure and help security issues that have been raised about these applications. The variety of policies implemented in this project may not showcase what can be fully achieved with this type of solution but they are there to give an indication of what could be done and how the solution and project itself can be taken further.

I now refer you to consider the appendices for further test evidence documented during this report.

9 References

[1] – Louis Columbus: Roundup of Internet of Things Forecasts and Market Estimates 2016, Forbes, 17/11/2016
Accessed 02/02/2017
http://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/#6450e4244ba5

[2] - Alex Drozhzhin: Internet of Crappy Things, 19/02/2015
Accessed: 05/02/2017
https://blog.kaspersky.com/internet-of-crappy-things/7667/

[3] - Andrew Meola: How the Internet of Things will affect security & privacy, Business Insider, 19/12/2016
Accessed: 05/02/2017
http://uk.businessinsider.com/internet-of-things-security-privacy-2016-8

[4] – Andrew Meola: How the Internet of Things will affect security & privacy, Business Insider, 19/12/2016
Accessed: 05/02/2017
http://uk.businessinsider.com/internet-of-things-security-privacy-2016-8

[5] – IFTTT, In the Beginning, 14/10/2014
Accessed: 06/02/2017
https://ifttt.com/blog/2010/12/ifttt-the-beginning

[6] – Vladimir Jirasek: IFTTT – A great service but should we trust it with our secrets?, 31/12/2011
Accessed: 02/02/2017
http://www.jirasekonsecurity.com/2011/12/ifttt-great-service-but-should-we-trust.html

[7] - Nitesh Dhanjani: Abusing the Internet of Things: Blackouts, Freakouts, and Stakeouts, O'Reilly, 2015
Accessed: 31/01/2017
Chapter 1, pages 32-36

[8] – Nick Peers: Your Online Life Made Simpler, Thanks to IFTTT, 02/10/2014
Accessed: /05/02/2017
http://blog.1and1.co.uk/2014/10/02/your-online-life-made-simpler-thanks-to-ifttt/

[9] – Rose Thibodeaux: Comparing IFTTT and Stringify, 06/01/2017
Accessed: 02/02/2017
http://homealarmreport.com/stringify-ifttt-hands-review-comparison/

[10] - Rose Thibodeaux: Comparing IFTTT and Stringify, 06/01/2017
Accessed: 02/02/2017
http://homealarmreport.com/stringify-ifttt-hands-review-comparison/

[11] – Microsoft Office 365 Trust Centre
Accessed: 10/02/2017
https://products.office.com/en-us/business/office-365-trust-center-welcome?legRedir=true&CorrelationId=20ac708c-b74f-4929-9c7f-03cd30ee1329

[12] – Sunay Vaishnav: Introducing Microsoft Flow Admin Center, 31/10/2016
Accessed: 10/02/2017
https://flow.microsoft.com/en-us/blog/intro-flow-admin-center/

[13] – Dweet.io homepage
Accessed: 10/03/2017
http://dweet.io/

[14] – ThingSpeak homepage
Accessed: 10/03/2017
https://thingspeak.com/

[15] – Alasdair Allan: The Maker Channel, 26/06/2015
Accessed: 17/03/2017
http://makezine.com/2015/06/26/ifttt-adds-new-channel-makers/

[16] – Simon Monk: Raspberry Pi Cookbook, O'Reilly, 2016
Accessed: 24/02/17
Chapter 15, page 430-433

[17] – Matt Richardson: Flask Web development, one drop at a time
Accessed: 16/02/2017
http://mattrichardson.com/Raspberry-Pi-Flask/index.html

[18] – Question on Quora, answered by Alexander Tibbets, Works for IFTTT, 06/09/2012
Accessed: 10/04/2017
https://www.quora.com/IFTTT-1/IFTTT-What-is-the-maximum-wait-time-for-a-trigger-to-occur-Not-quick-triggers

Figure References 9.1

Figure 2.2 – Example Applet from IFTTT
Accessed: 02/02/2017
https://ifttt.com/applets/54681p-when-you-re-tagged-in-a-facebook-photo-save-it-to-google-drive

Figure 2.3 – Services offered on IFTTT
Accessed 02/02/2017
https://ifttt.com/search/services

Figure 2.4 – Stringify Home page
Accessed: 03/02/2017
https://www.stringify.com/

Figure 2.5 – Microsoft Flow services available
Accessed: 03/02/2017
https://flow.microsoft.com/en-us/services/

Figure 2.6 – Dweet.io homepage
Accessed: 10/03/2017
http://dweet.io/

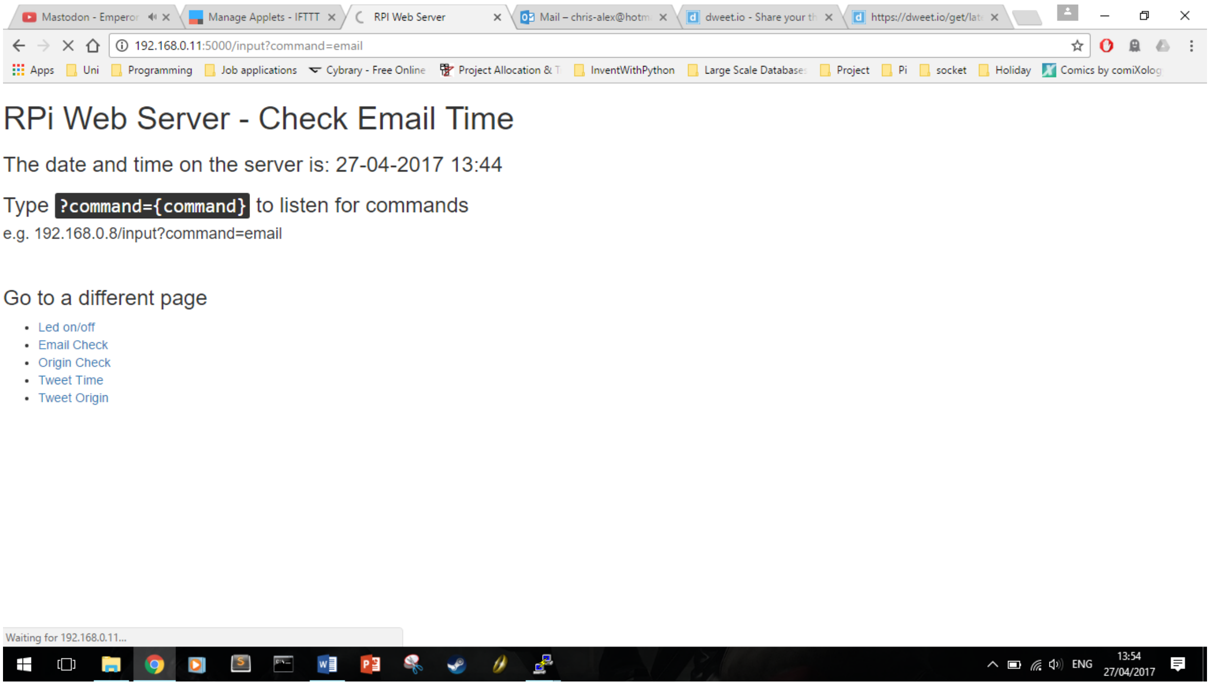
10 Appendices

10.1 Table of Abbreviations

Abbreviation	**Word**
IoT	Internet of Things
IFTTT	If-This-Then-That
GPIO	General Purpose Input/Output
OAuth	Open Authentication
SSL	Secure Socket Layer
DLP	Data Loss Prevention
LED	Light Emitting Diode
API	Application Programming Interface
IP	Internet Protocol

10.2 Testing Screenshots

10.2.1 Test number 1



RPi Web Server - Check Email Time

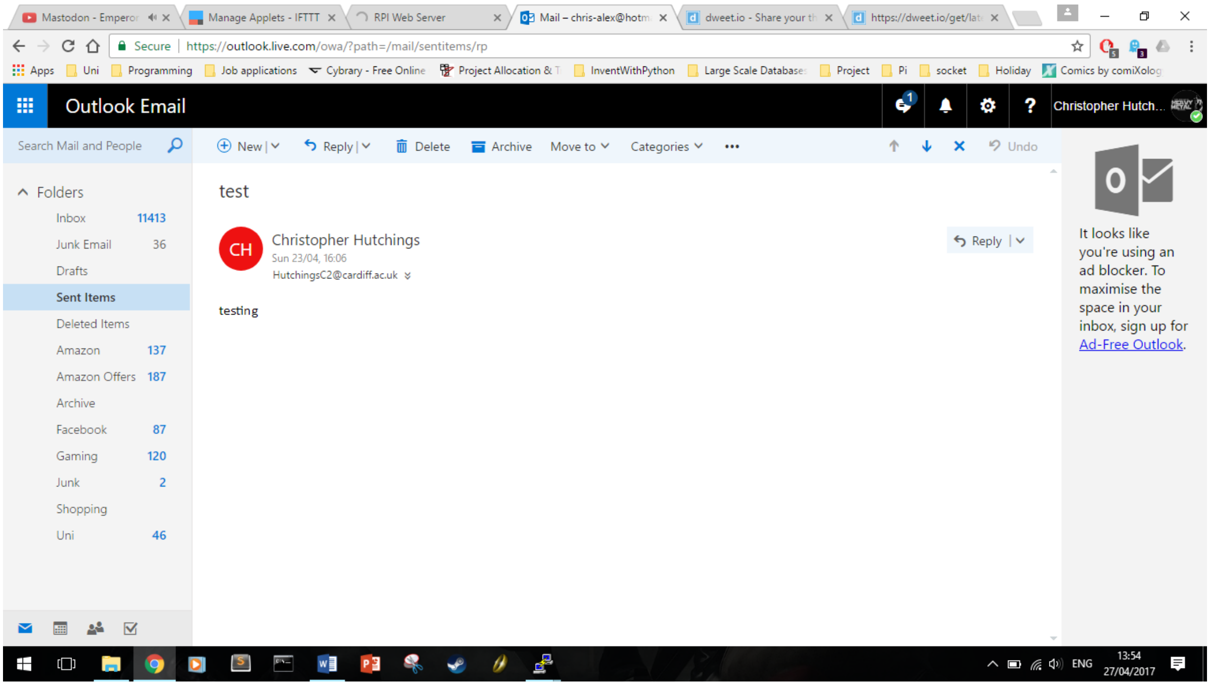
The date and time on the server is: 27-04-2017 13:44

Type `?command={command}` to listen for commands

e.g. 192.168.0.8/input?command=email

Go to a different page

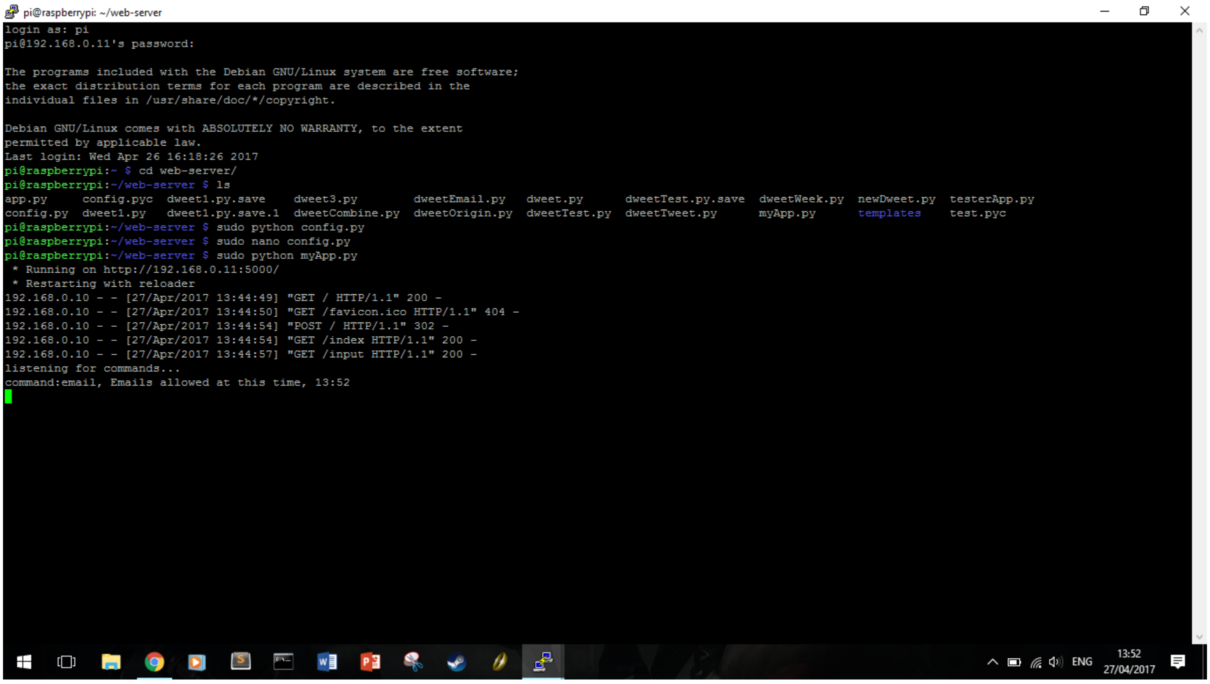
- Led on/off
- Email Check
- Origin Check
- Tweet Time
- Tweet Origin



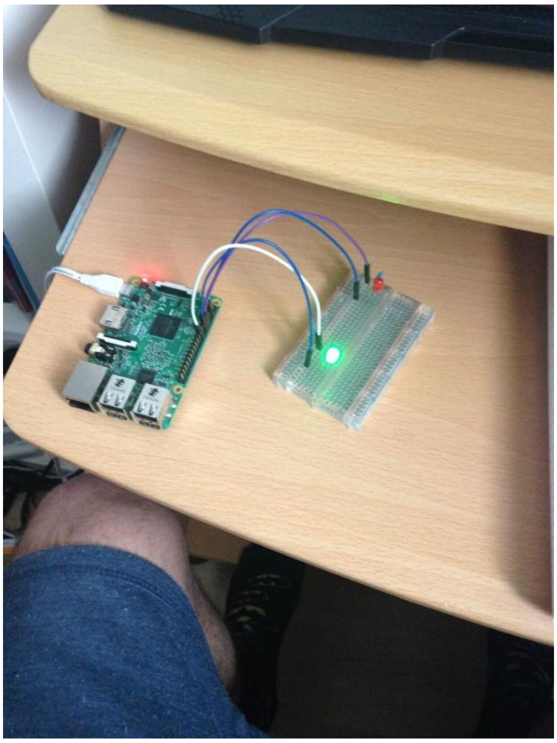
test

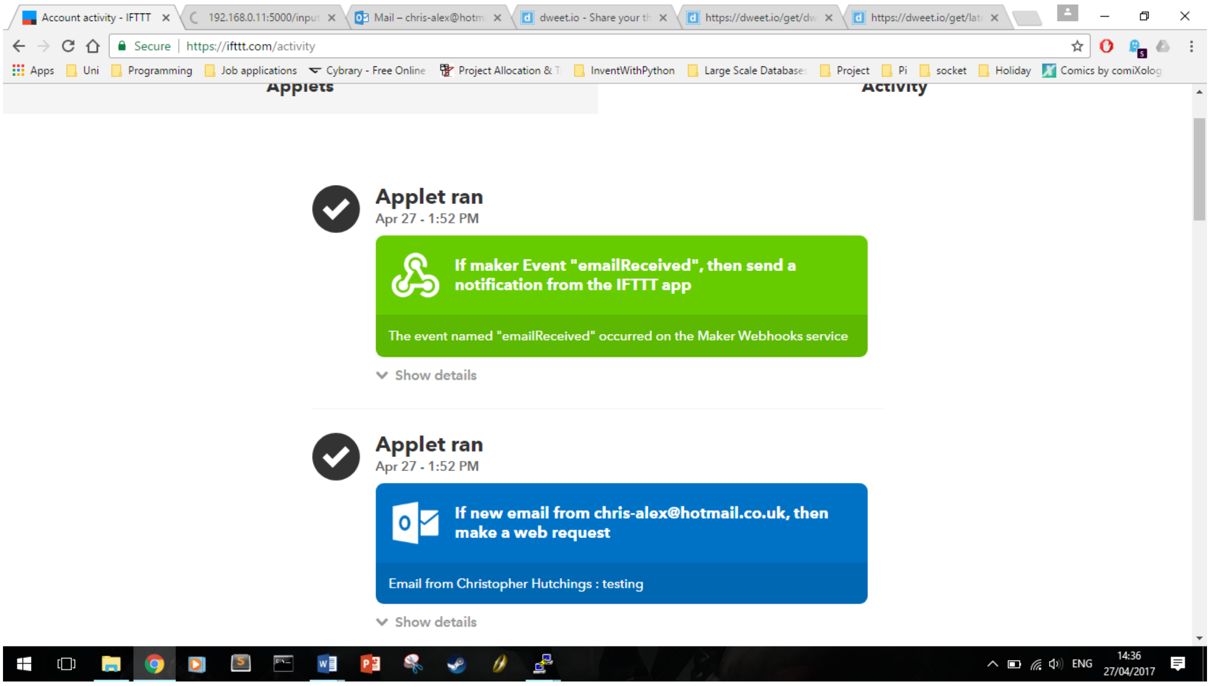
CH Christopher Hutchings
Sun 23/04, 16:06
HutchingsC2@cardiff.ac.uk

testing



```
login as: pi
pi@192.168.0.11's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Apr 26 16:18:26 2017
pi@raspberrypi:~ $ cd web-server/
pi@raspberrypi:~/web-server $ ls
app.py     config.pyc  dweet1.py.save    dweet3.py        dweetEmail.py   dweet.py      dweetTest.py.save  dweetWeek.py  newDweet.py  testerApp.py
config.py  dweet1.py   dweet1.py.save.1  dweetCombine.py  dweetOrigin.py  dweetTest.py  dweetTweet.py      myApp.py      templates    test.pyc
pi@raspberrypi:~/web-server $ sudo python config.py
pi@raspberrypi:~/web-server $ sudo nano config.py
pi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
192.168.0.10 - - [27/Apr/2017 13:44:49] "GET / HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 13:44:50] "GET /favicon.ico HTTP/1.1" 404 -
192.168.0.10 - - [27/Apr/2017 13:44:54] "POST / HTTP/1.1" 302 -
192.168.0.10 - - [27/Apr/2017 13:44:54] "GET /index HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 13:44:57] "GET /input HTTP/1.1" 200 -
listening for commands...
command:email, Emails allowed at this time, 13:52
```





Applet ran
Apr 27 - 1:52 PM

If maker Event "emailReceived", then send a notification from the IFTTT app

The event named "emailReceived" occurred on the Maker Webhooks service

Show details

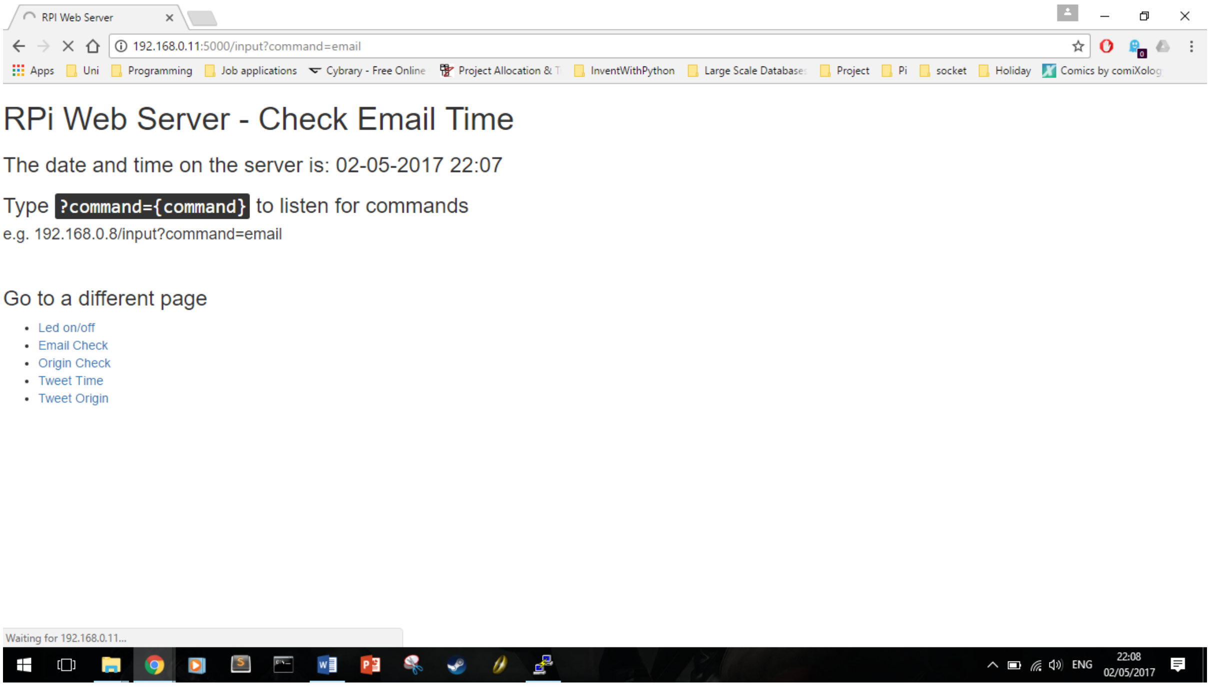
Applet ran
Apr 27 - 1:52 PM

If new email from chris-alex@hotmail.co.uk, then make a web request

Email from Christopher Hutchings : testing

Show details

10.2.2 Test number 2



RPi Web Server
192.168.0.11:5000/input?command=email
Waiting for 192.168.0.11...

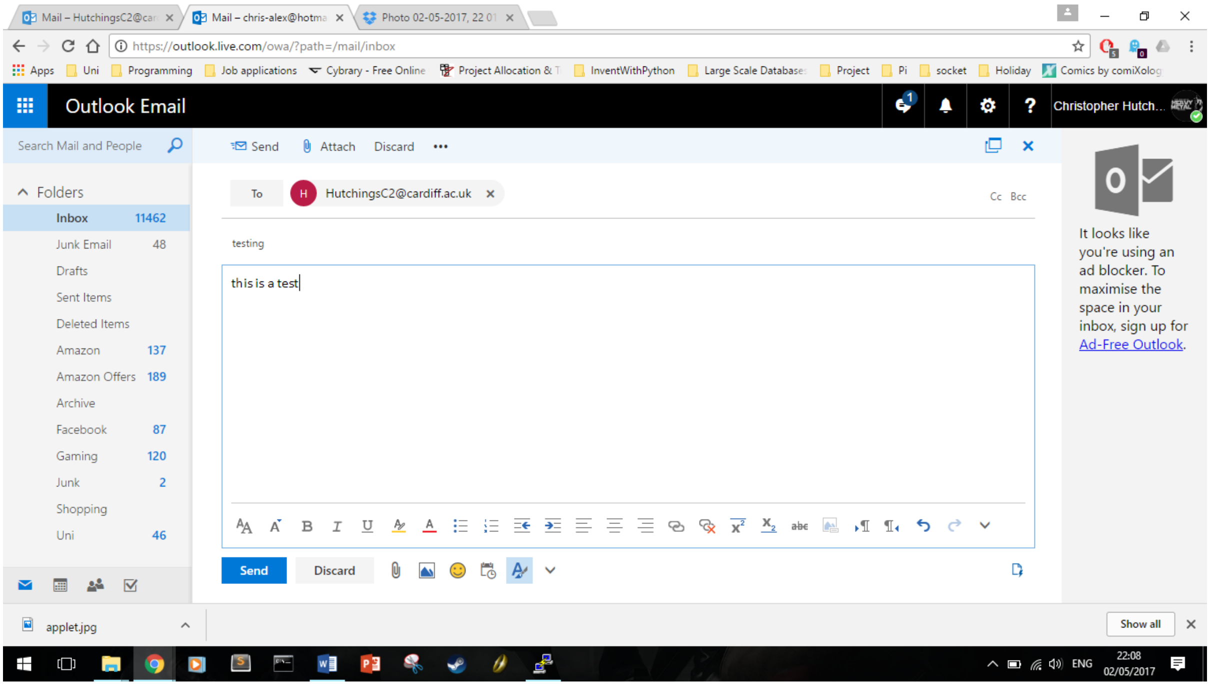
RPi Web Server - Check Email Time

The date and time on the server is: 02-05-2017 22:07

Type `?command={command}` to listen for commands
e.g. 192.168.0.8/input?command=email

Go to a different page

- Led on/off
- Email Check
- Origin Check
- Tweet Time
- Tweet Origin

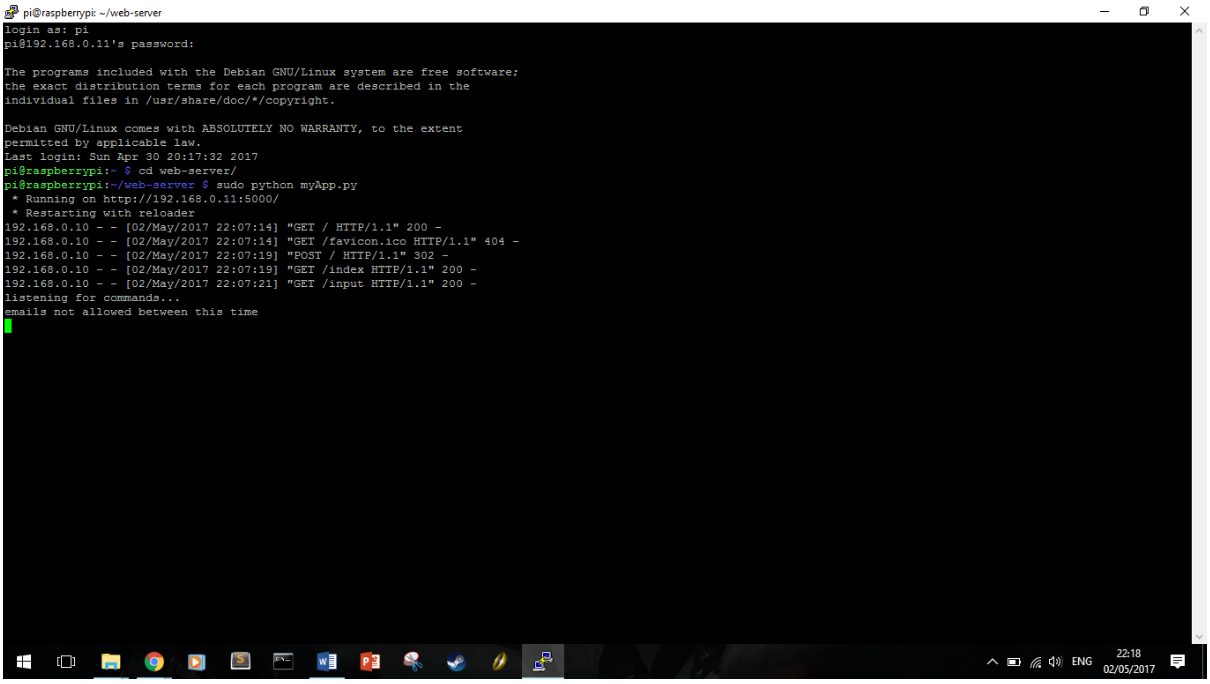


To HutchingsC2@cardiff.ac.uk

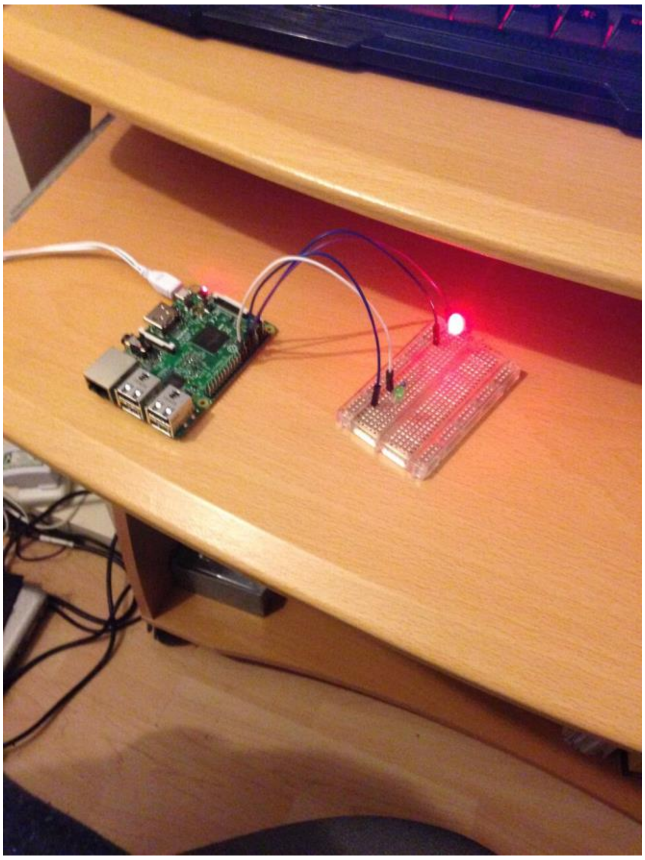
testing

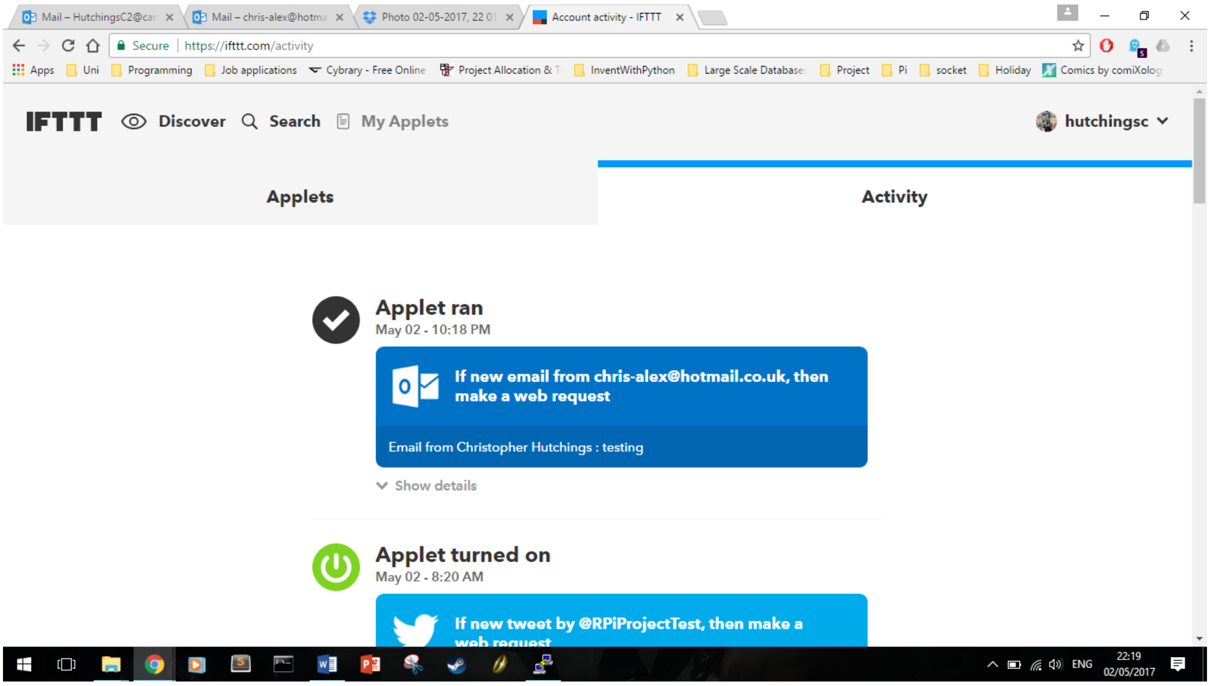
this is a test

It looks like you're using an ad blocker. To maximise the space in your inbox, sign up for Ad-Free Outlook.



```
login as: pi
pi@192.168.0.11's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Apr 30 20:17:32 2017
pi@raspberrypi:~ $ cd web-server/
pi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
192.168.0.10 - - [02/May/2017 22:07:14] "GET / HTTP/1.1" 200 -
192.168.0.10 - - [02/May/2017 22:07:14] "GET /favicon.ico HTTP/1.1" 404 -
192.168.0.10 - - [02/May/2017 22:07:19] "POST / HTTP/1.1" 302 -
192.168.0.10 - - [02/May/2017 22:07:19] "GET /index HTTP/1.1" 200 -
192.168.0.10 - - [02/May/2017 22:07:21] "GET /input HTTP/1.1" 200 -
listening for commands...
emails not allowed between this time
```





Applet ran
May 02 - 10:18 PM

If new email from chris-alex@hotmail.co.uk, then make a web request

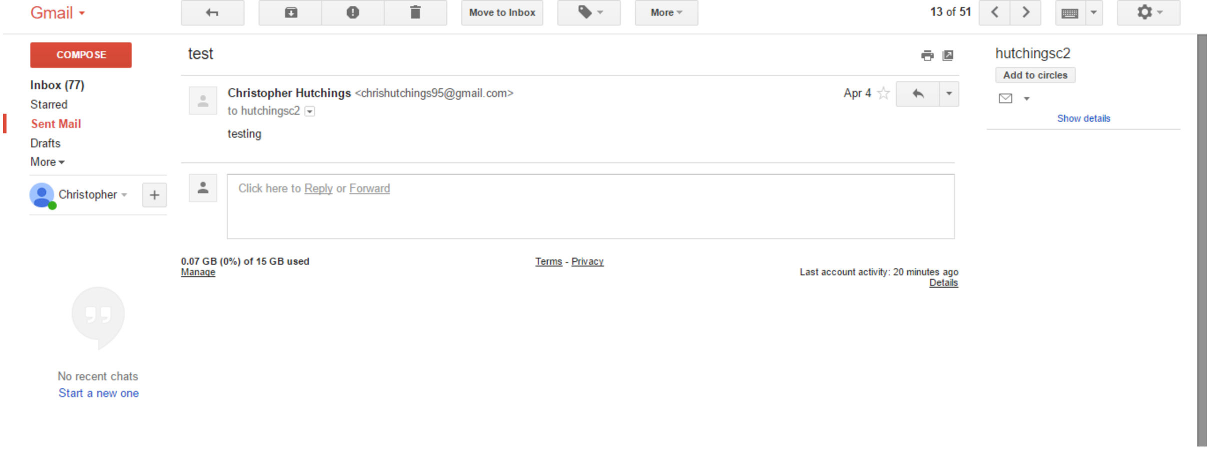
Email from Christopher Hutchings : testing

Show details

Applet turned on
May 02 - 8:20 AM

If new tweet by @RPiProjectTest, then make a web request

10.2.3 Test number 3



test

Christopher Hutchings <chrishutchings95@gmail.com>
to hutchingsc2
Apr 4

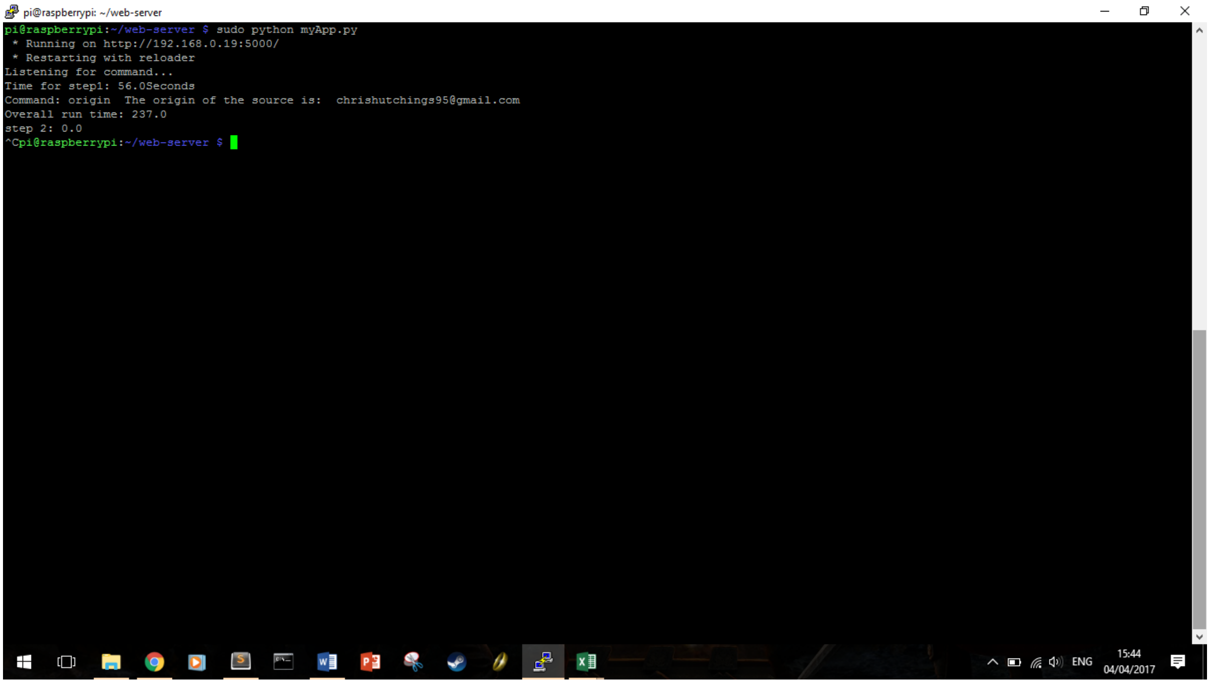
testing

Click here to Reply or Forward

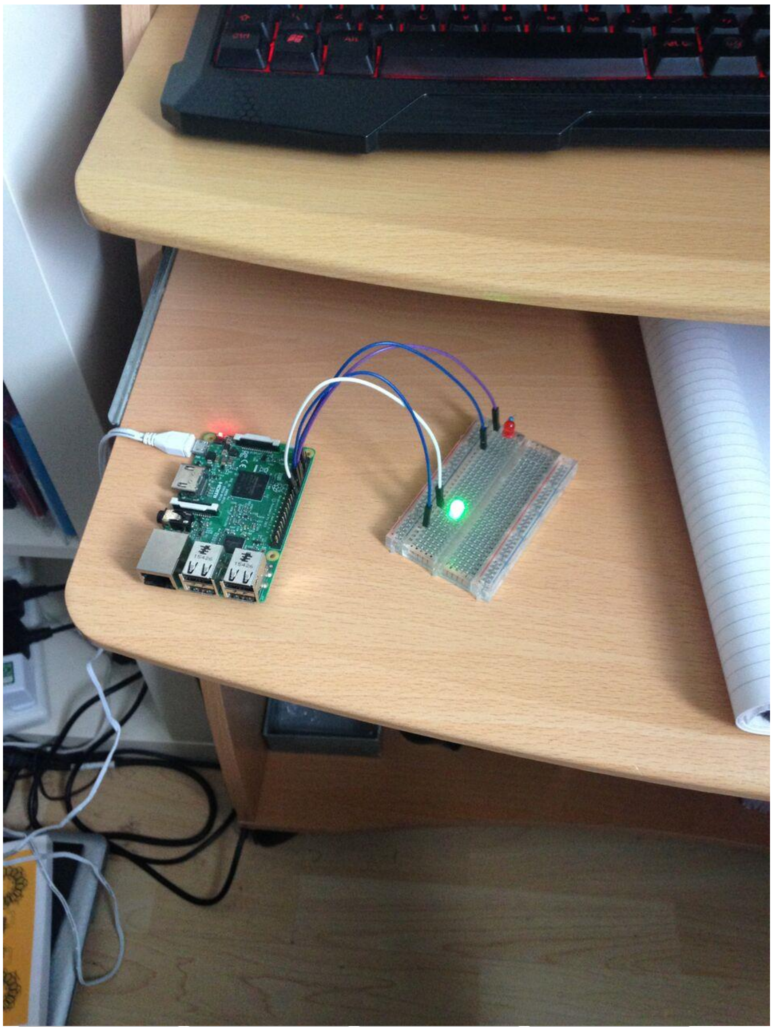
0.07 GB (0%) of 15 GB used
Manage

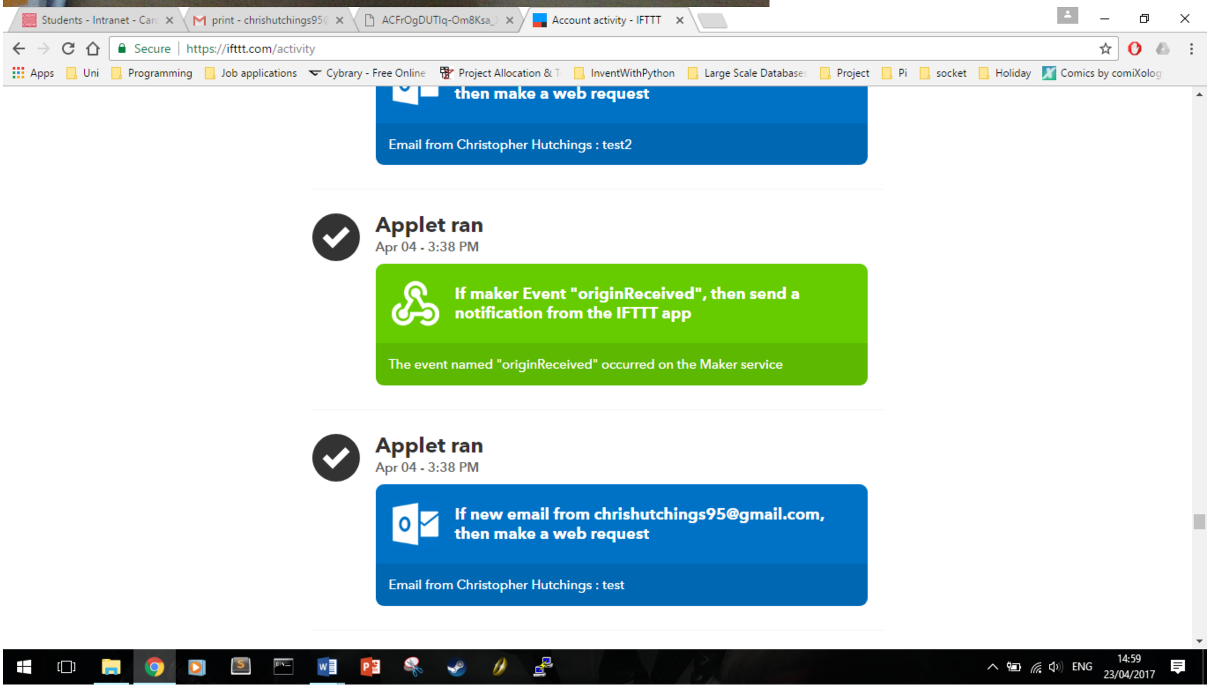
Terms - Privacy

Last account activity: 20 minutes ago
Details



```
pi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.19:5000/
 * Restarting with reloader
Listening for command...
Time for step1: 56.0Seconds
Command: origin  The origin of the source is:  chrishutchings95@gmail.com
Overall run time: 237.0
step 2: 0.0
^Cpi@raspberrypi:~/web-server $
```





then make a web request

Email from Christopher Hutchings : test2

Applet ran
Apr 04 - 3:38 PM

If maker Event "originReceived", then send a notification from the IFTTT app

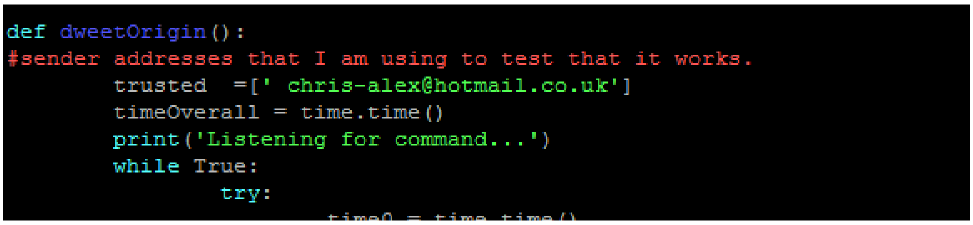
The event named "originReceived" occurred on the Maker service

Applet ran
Apr 04 - 3:38 PM

If new email from chrishutchings95@gmail.com, then make a web request

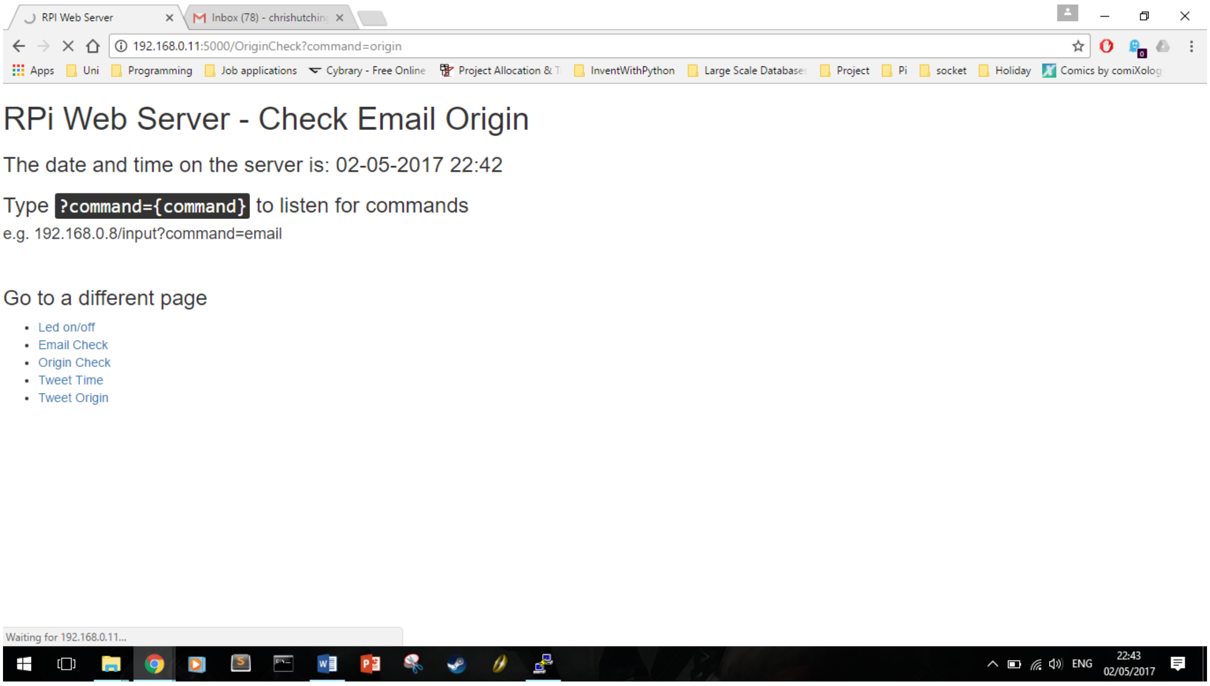
Email from Christopher Hutchings : test

10.2.4 Test number 4



```
def dweetOrigin():
#sender addresses that I am using to test that it works.
        trusted  =[' chris-alex@hotmail.co.uk']
        timeOverall = time.time()
        print('Listening for command...')
        while True:
                try:
                        time0 = time.time()
```

Email address not in the trusted list in the configuration file



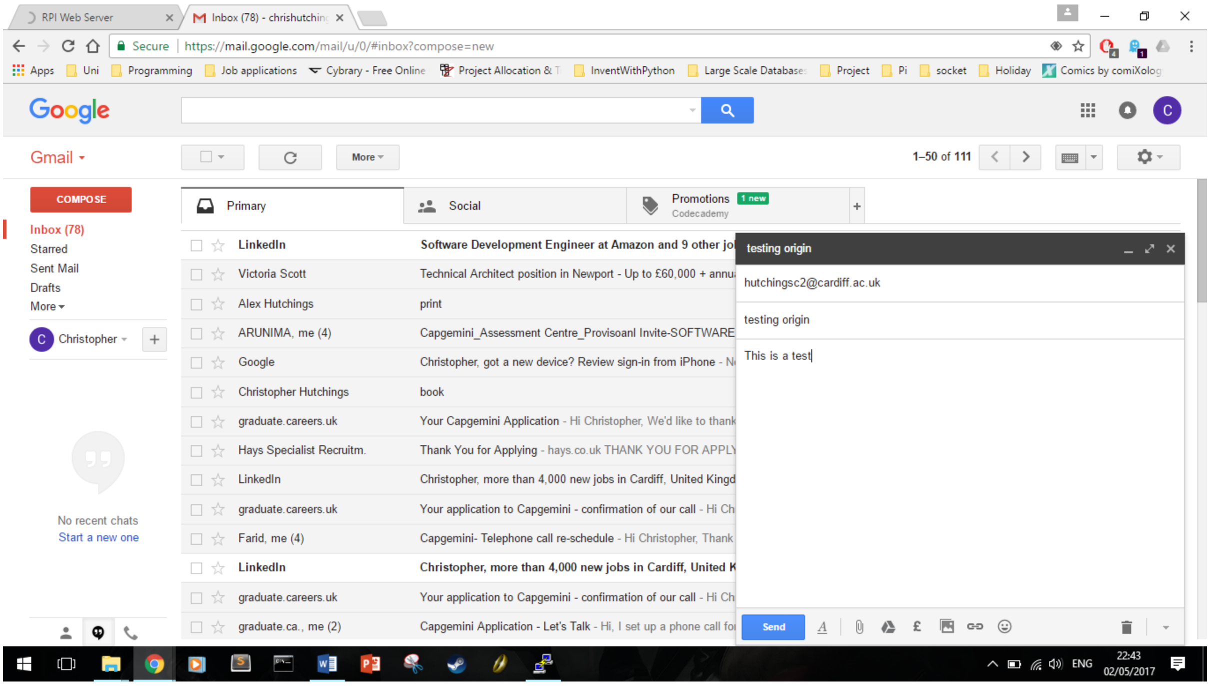
RPi Web Server - Check Email Origin

The date and time on the server is: 02-05-2017 22:42

Type `?command={command}` to listen for commands
e.g. 192.168.0.8/input?command=email

Go to a different page

- Led on/off
- Email Check
- Origin Check
- Tweet Time
- Tweet Origin



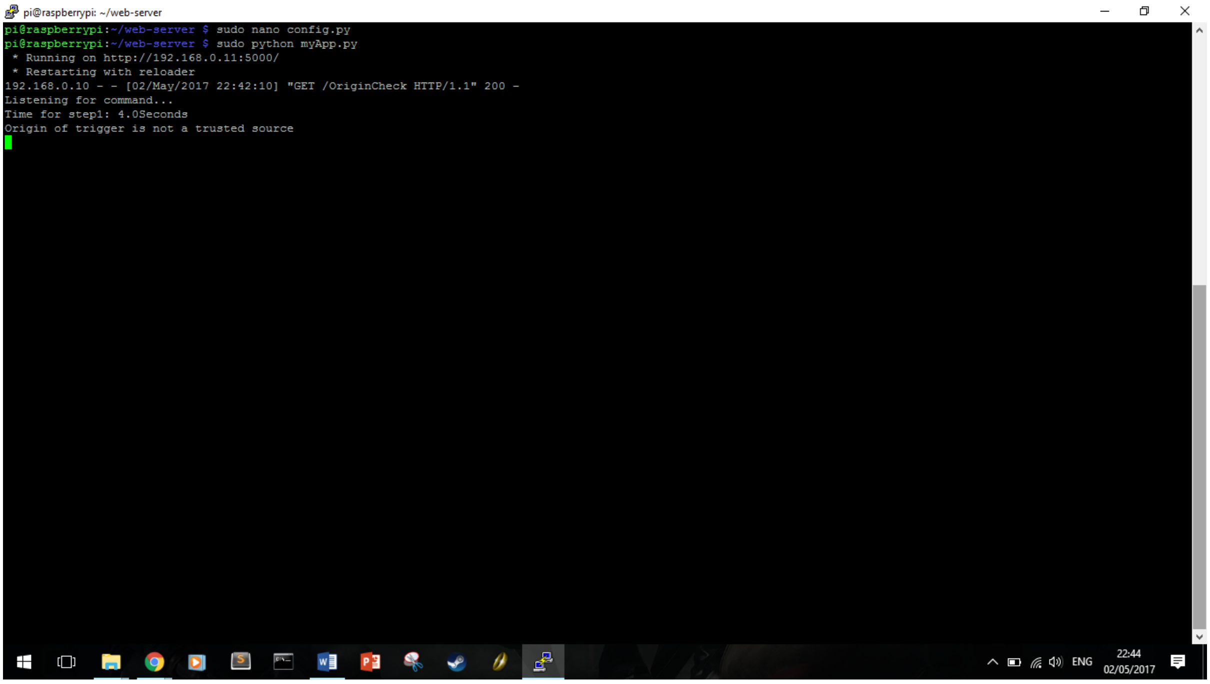
LinkedIn	Software Development Engineer at Amazon and 9 other jo
Victoria Scott	Technical Architect position in Newport - Up to £60,000 + annu
Alex Hutchings	print
ARUNIMA, me (4)	Capgemini_Assessment Centre_Provisoanl Invite-SOFTWARE
Google	Christopher, got a new device? Review sign-in from iPhone - N
Christopher Hutchings	book
graduate.careers.uk	Your Capgemini Application - Hi Christopher, We'd like to thank
Hays Specialist Recruitm.	Thank You for Applying - hays.co.uk THANK YOU FOR APPLY
LinkedIn	Christopher, more than 4,000 new jobs in Cardiff, United Kingd
graduate.careers.uk	Your application to Capgemini - confirmation of our call - Hi Ch
Farid, me (4)	Capgemini- Telephone call re-schedule - Hi Christopher, Thank
LinkedIn	Christopher, more than 4,000 new jobs in Cardiff, United K
graduate.careers.uk	Your application to Capgemini - confirmation of our call - Hi Ch
graduate.ca., me (2)	Capgemini Application - Let's Talk - Hi, I set up a phone call fo

testing origin

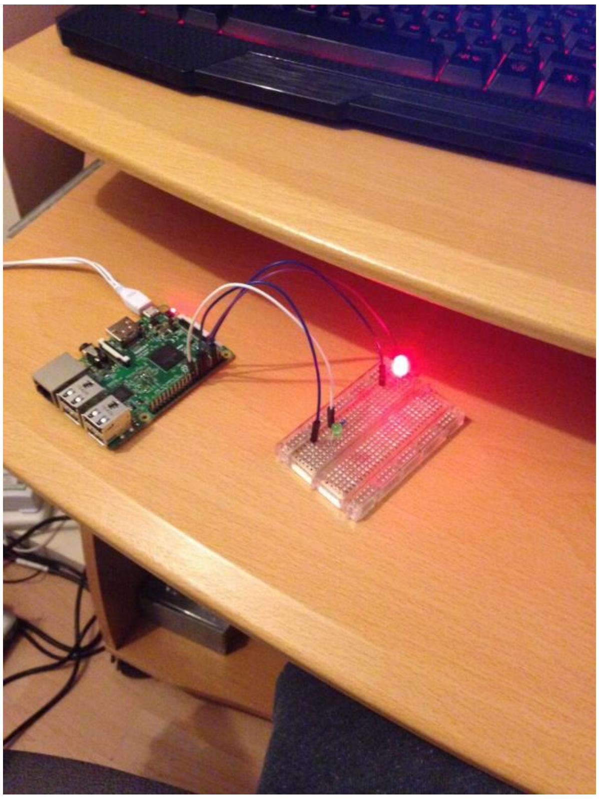
hutchingsc2@cardiff.ac.uk

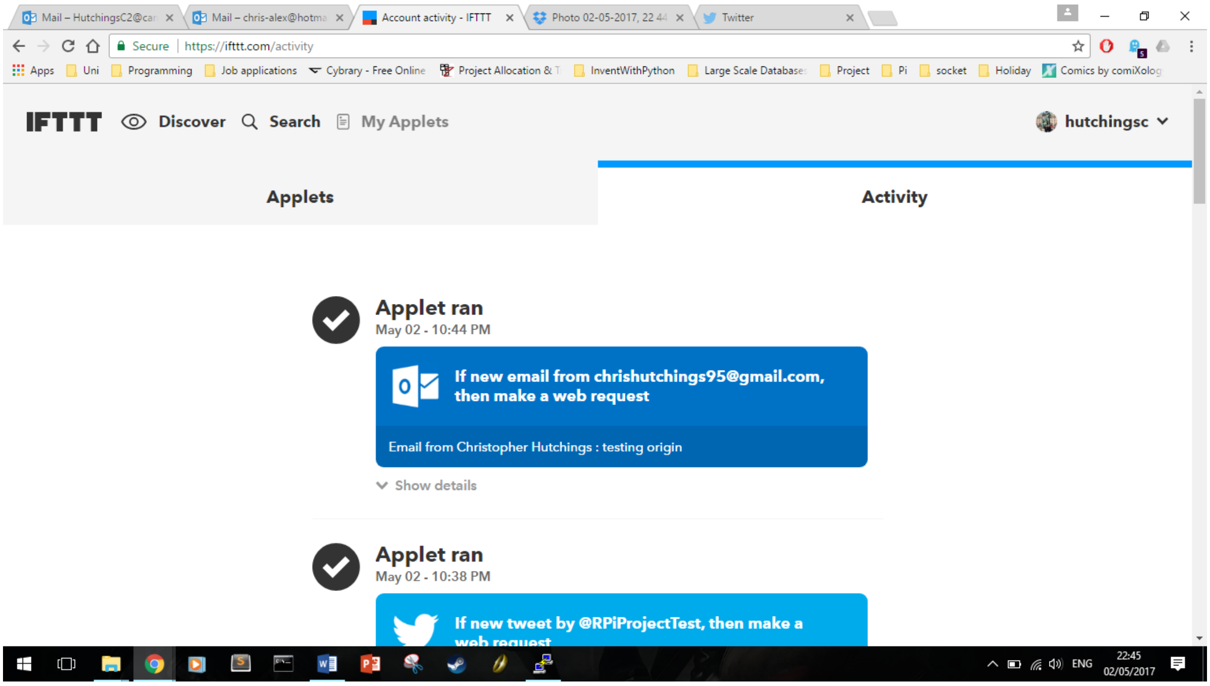
testing origin

This is a test



```
pi@raspberrypi:~/web-server $ sudo nano config.py
pi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
192.168.0.10 - - [02/May/2017 22:42:10] "GET /OriginCheck HTTP/1.1" 200 -
Listening for command...
Time for step1: 4.0Seconds
Origin of trigger is not a trusted source
```





Applet ran

May 02 - 10:44 PM

If new email from chrishutchings95@gmail.com, then make a web request

Email from Christopher Hutchings : testing origin

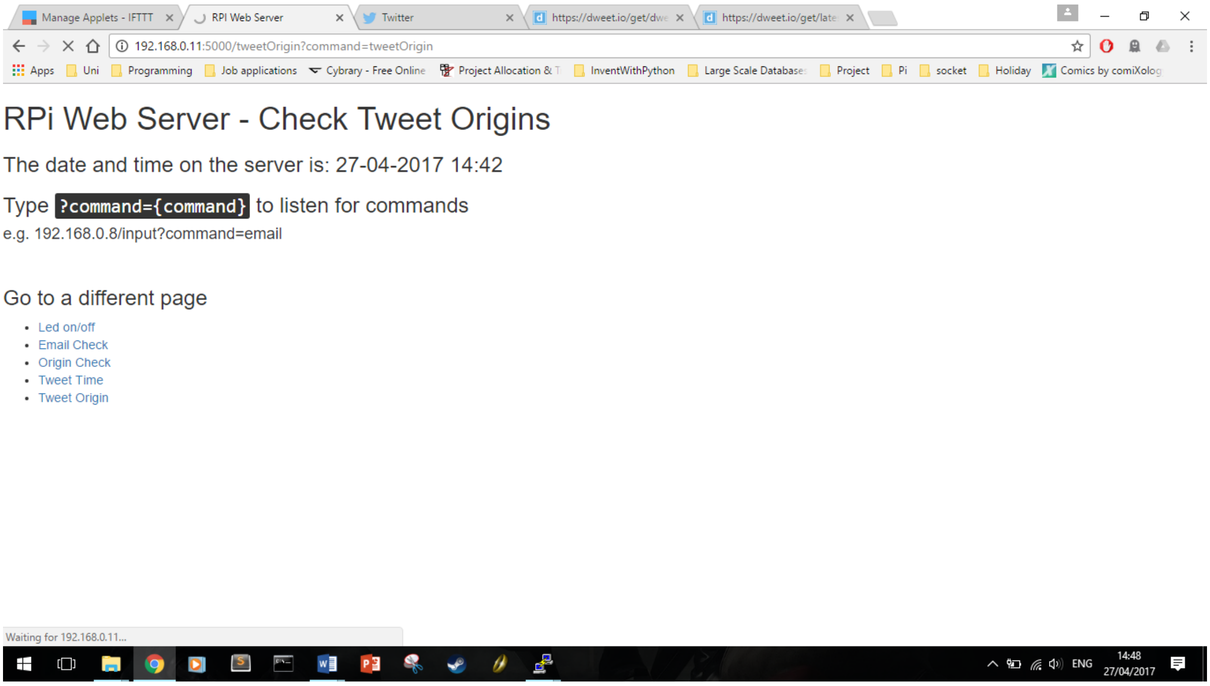
Show details

Applet ran

May 02 - 10:38 PM

If new tweet by @RPiProjectTest, then make a web request

10.2.5 Test number 5



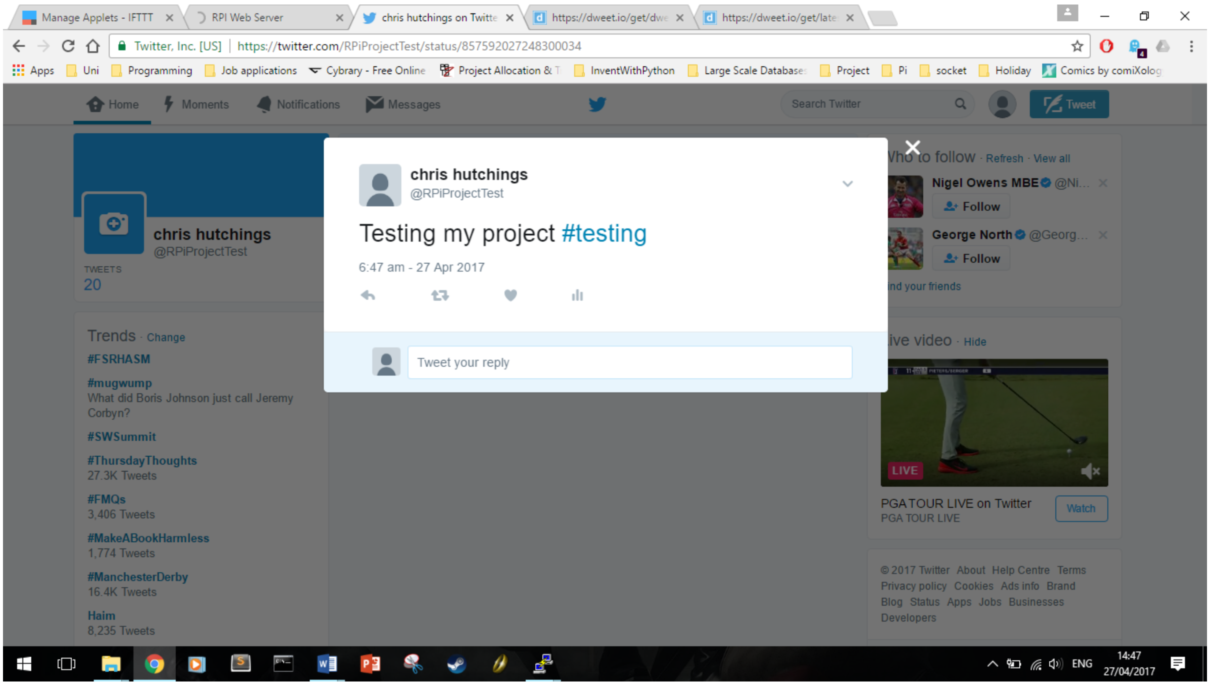
RPi Web Server - Check Tweet Origins

The date and time on the server is: 27-04-2017 14:42

Type `?command={command}` to listen for commands
e.g. 192.168.0.8/input?command=email

Go to a different page

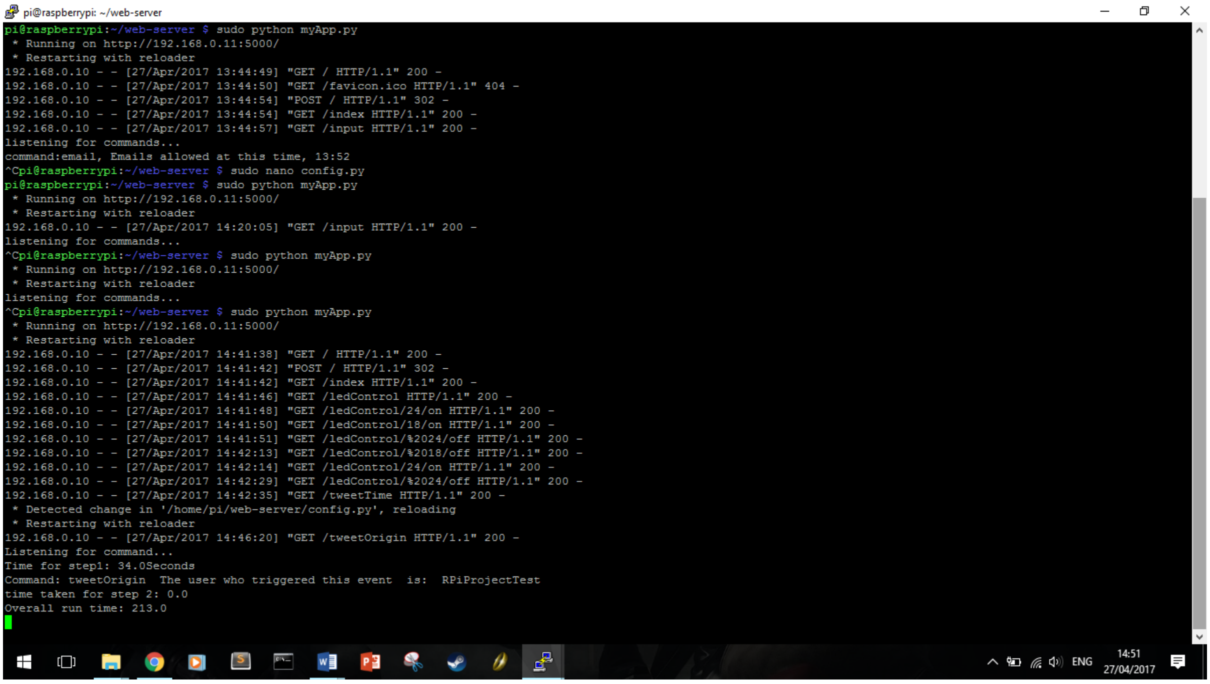
- Led on/off
- Email Check
- Origin Check
- Tweet Time
- Tweet Origin



chris hutchings
@RPiProjectTest

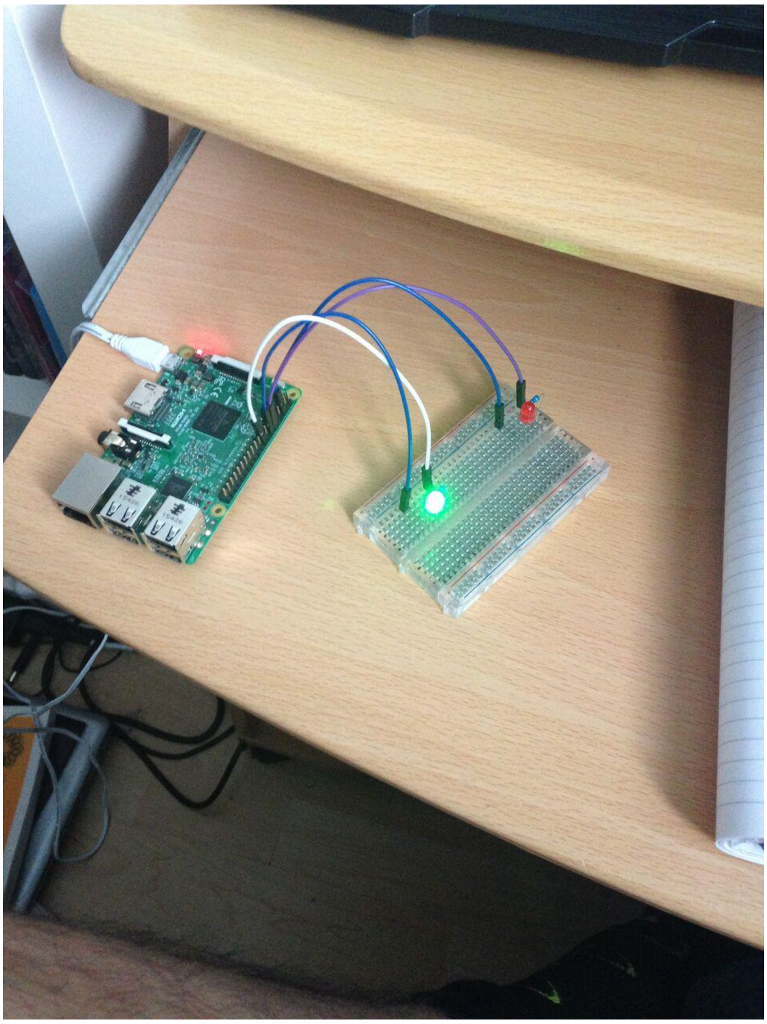
Testing my project #testing

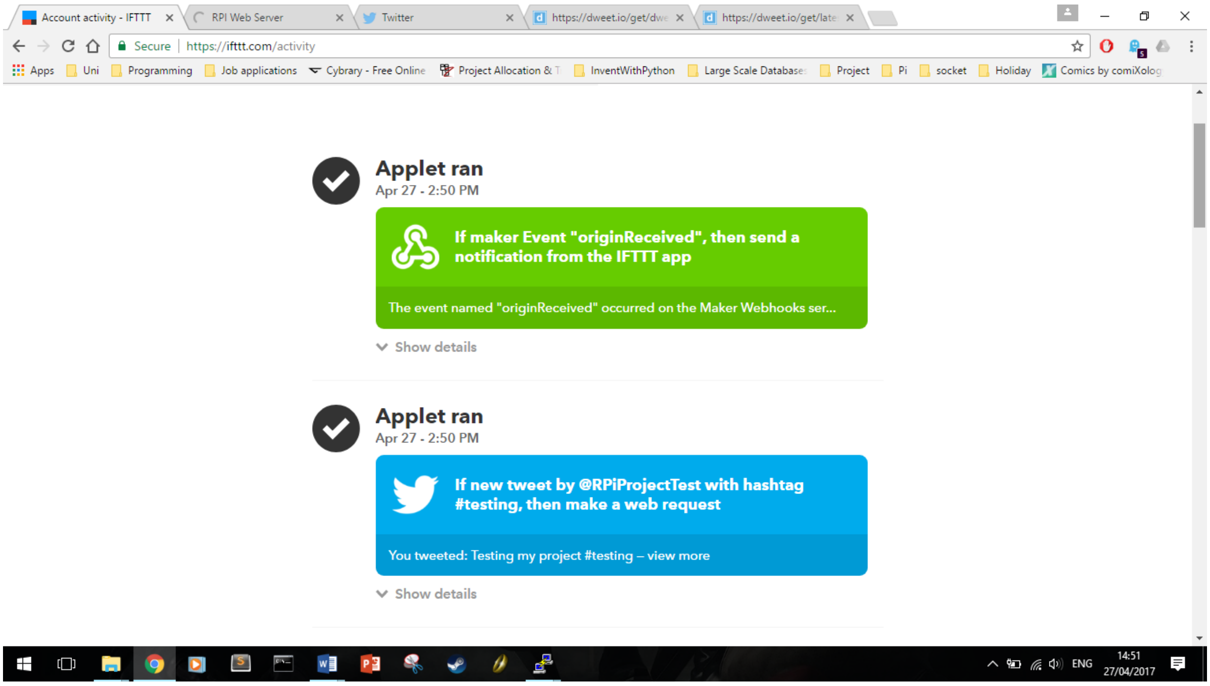
6:47 am - 27 Apr 2017



pi@raspberrypi: ~/web-server

```
pi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
192.168.0.10 - - [27/Apr/2017 13:44:49] "GET / HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 13:44:50] "GET /favicon.ico HTTP/1.1" 404 -
192.168.0.10 - - [27/Apr/2017 13:44:54] "POST / HTTP/1.1" 302 -
192.168.0.10 - - [27/Apr/2017 13:44:54] "GET /index HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 13:44:57] "GET /input HTTP/1.1" 200 -
listening for commands...
command:email, Emails allowed at this time, 13:52
^Cpi@raspberrypi:~/web-server $ sudo nano config.py
pi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
192.168.0.10 - - [27/Apr/2017 14:20:05] "GET /input HTTP/1.1" 200 -
listening for commands...
^Cpi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
listening for commands...
^Cpi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
192.168.0.10 - - [27/Apr/2017 14:41:38] "GET / HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:41:42] "POST / HTTP/1.1" 302 -
192.168.0.10 - - [27/Apr/2017 14:41:42] "GET /index HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:41:46] "GET /ledControl HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:41:48] "GET /ledControl/24/on HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:41:50] "GET /ledControl/18/on HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:41:51] "GET /ledControl/%2024/off HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:42:13] "GET /ledControl/%2018/off HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:42:14] "GET /ledControl/24/on HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:42:29] "GET /ledControl/%2024/off HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:42:35] "GET /tweetTime HTTP/1.1" 200 -
 * Detected change in '/home/pi/web-server/config.py', reloading
 * Restarting with reloader
192.168.0.10 - - [27/Apr/2017 14:46:20] "GET /tweetOrigin HTTP/1.1" 200 -
Listening for command...
Time for step1: 34.0Seconds
Command: tweetOrigin  The user who triggered this event  is:  RPiProjectTest
time taken for step 2: 0.0
Overall run time: 213.0
```





Applet ran
Apr 27 - 2:50 PM

If maker Event "originReceived", then send a notification from the IFTTT app

The event named "originReceived" occurred on the Maker Webhooks ser...

Show details

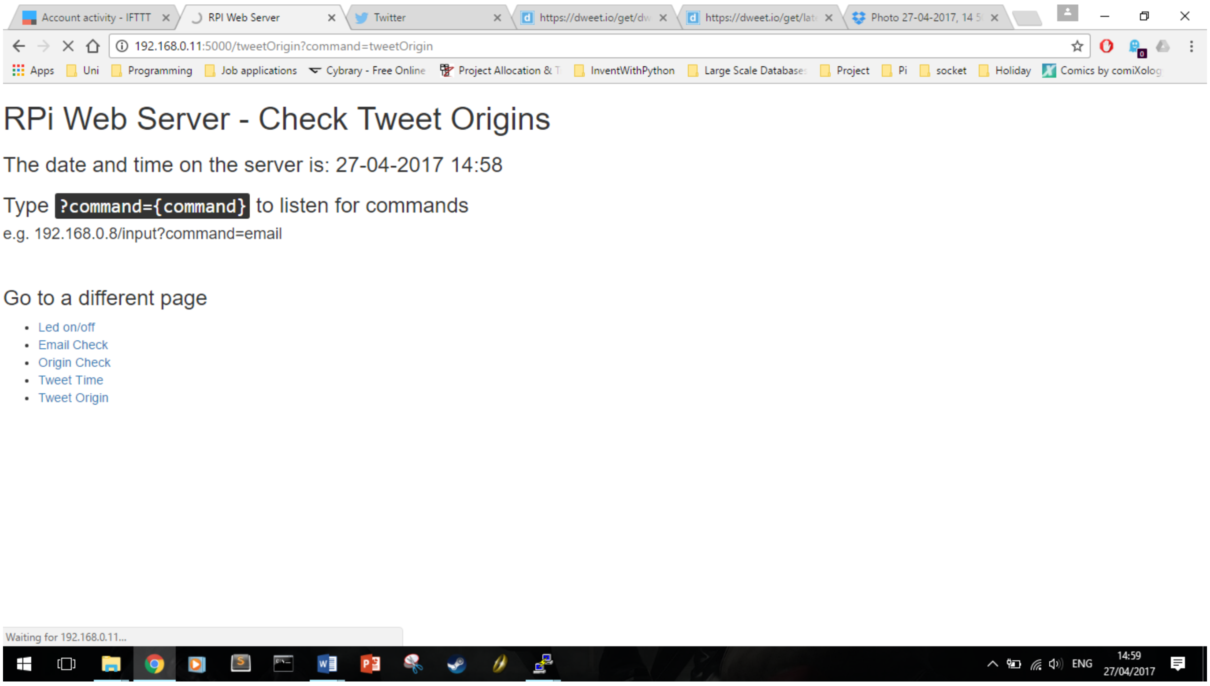
Applet ran
Apr 27 - 2:50 PM

If new tweet by @RPiProjectTest with hashtag #testing, then make a web request

You tweeted: Testing my project #testing – view more

Show details

10.2.6 Test number 6



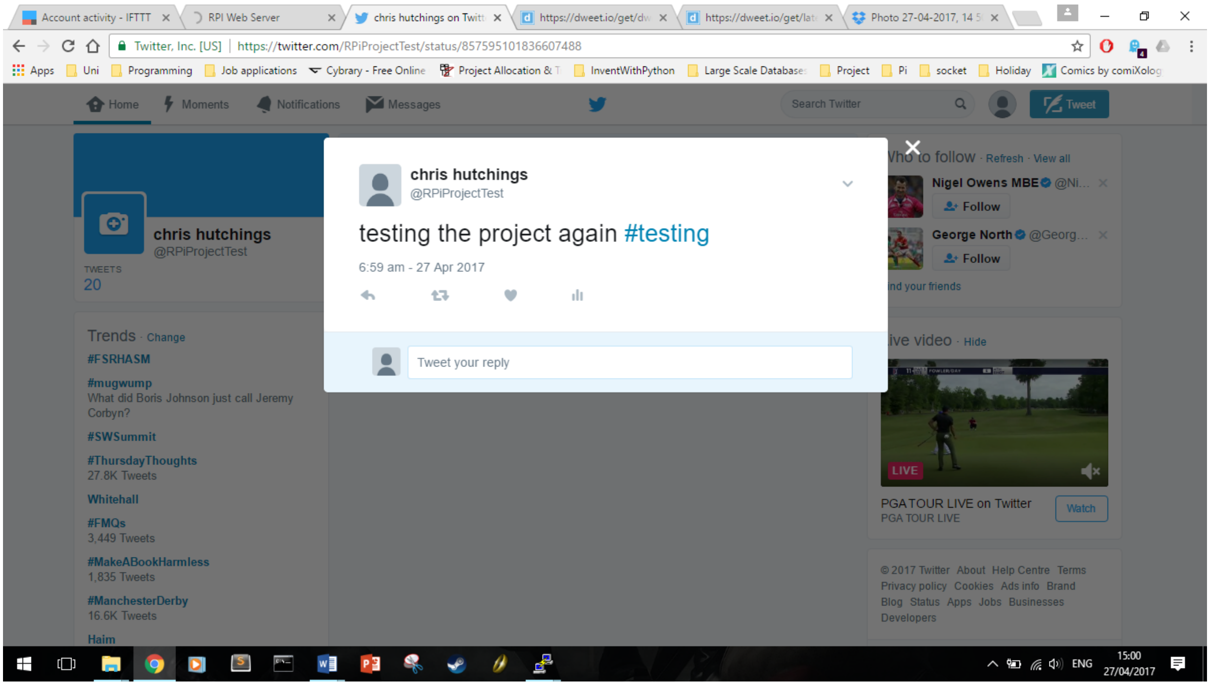
RPi Web Server - Check Tweet Origins

The date and time on the server is: 27-04-2017 14:58

Type `?command={command}` to listen for commands
e.g. 192.168.0.8/input?command=email

Go to a different page

- Led on/off
- Email Check
- Origin Check
- Tweet Time
- Tweet Origin

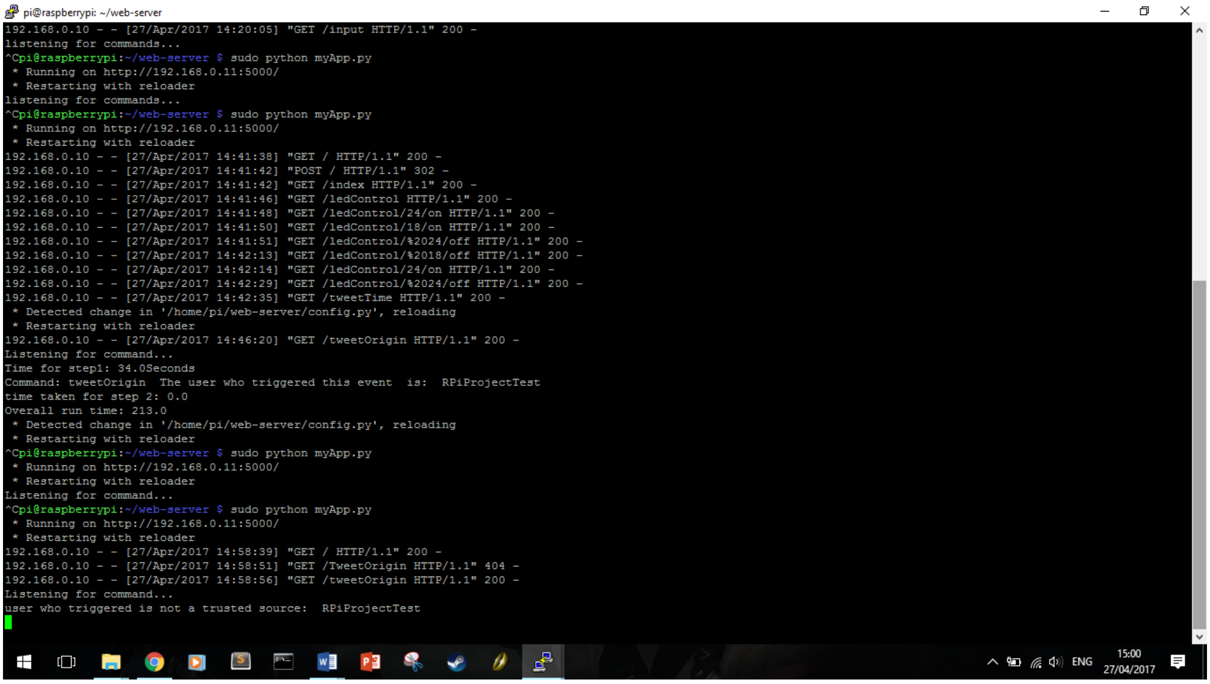


https://twitter.com/RPiProjectTest/status/857595101836607488

chris hutchings
@RPiProjectTest

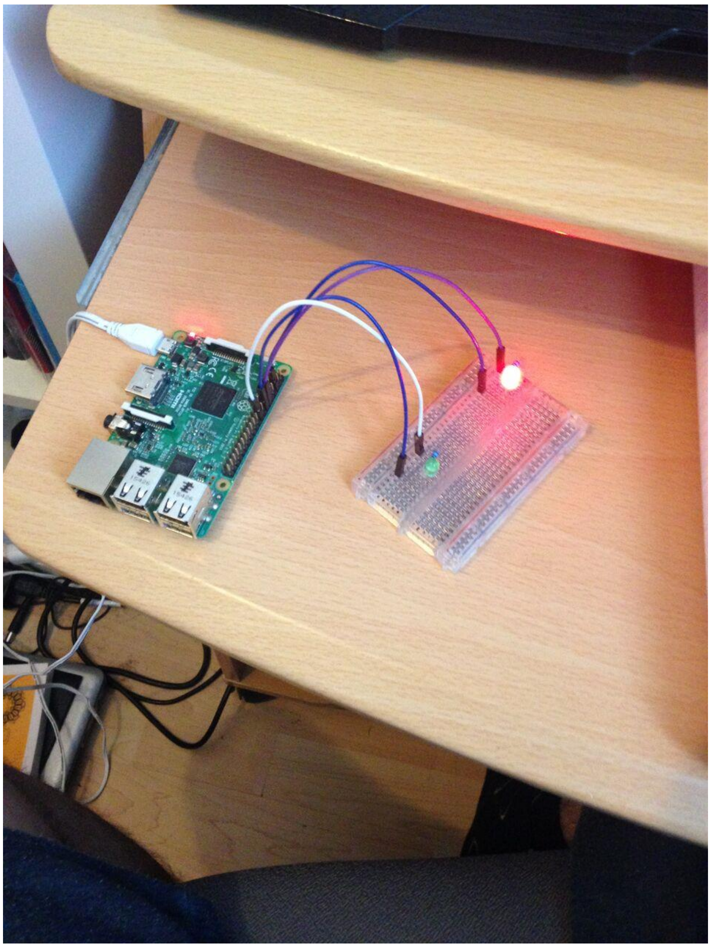
testing the project again #testing

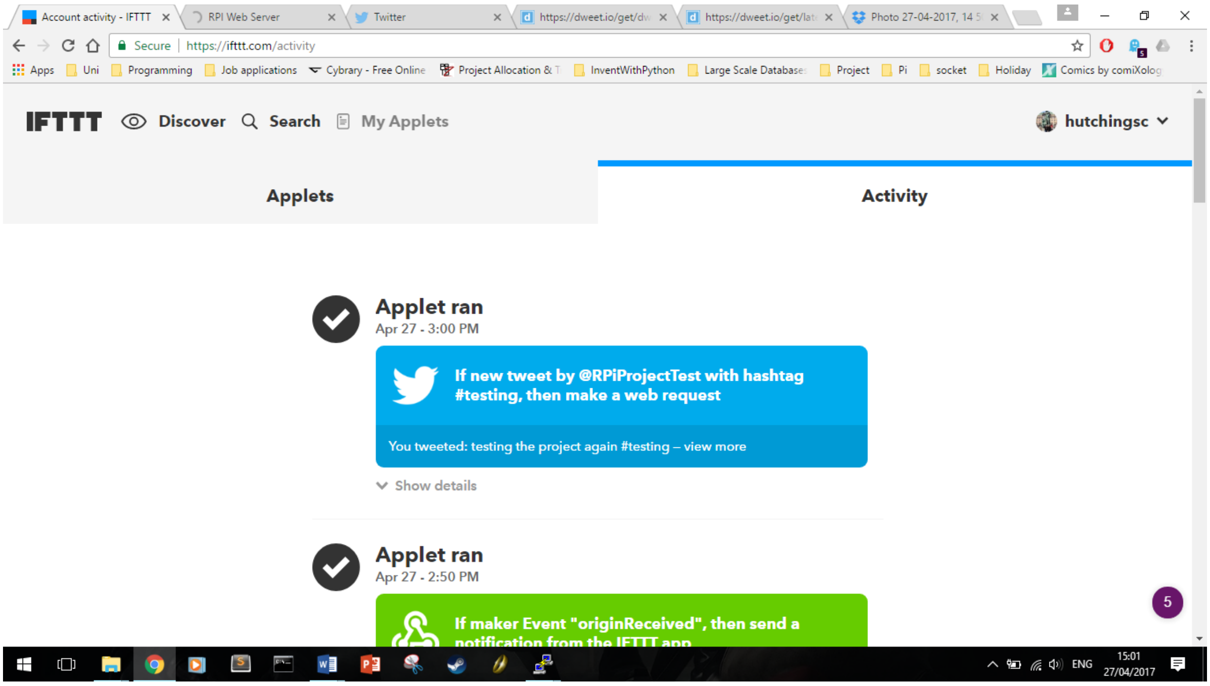
6:59 am - 27 Apr 2017



pi@raspberrypi: ~/web-server

```
192.168.0.10 - - [27/Apr/2017 14:20:05] "GET /input HTTP/1.1" 200 -
listening for commands...
^Cpi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
listening for commands...
^Cpi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
192.168.0.10 - - [27/Apr/2017 14:41:38] "GET / HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:41:42] "POST / HTTP/1.1" 302 -
192.168.0.10 - - [27/Apr/2017 14:41:42] "GET /index HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:41:46] "GET /ledControl HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:41:48] "GET /ledControl/24/on HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:41:50] "GET /ledControl/18/on HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:41:51] "GET /ledControl/%2024/off HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:42:13] "GET /ledControl/%2018/off HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:42:14] "GET /ledControl/24/on HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:42:29] "GET /ledControl/%2024/off HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:42:35] "GET /tweetTime HTTP/1.1" 200 -
 * Detected change in '/home/pi/web-server/config.py', reloading
 * Restarting with reloader
192.168.0.10 - - [27/Apr/2017 14:46:20] "GET /tweetOrigin HTTP/1.1" 200 -
Listening for command...
Time for step1: 34.0Seconds
Command: tweetOrigin  The user who triggered this event  is:  RPiProjectTest
time taken for step 2: 0.0
Overall run time: 213.0
 * Detected change in '/home/pi/web-server/config.py', reloading
 * Restarting with reloader
^Cpi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
Listening for command...
^Cpi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
192.168.0.10 - - [27/Apr/2017 14:58:39] "GET / HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 14:58:51] "GET /TweetOrigin HTTP/1.1" 404 -
192.168.0.10 - - [27/Apr/2017 14:58:56] "GET /tweetOrigin HTTP/1.1" 200 -
Listening for command...
user who triggered is not a trusted source:  RPiProjectTest
```





Applet ran
Apr 27 - 3:00 PM

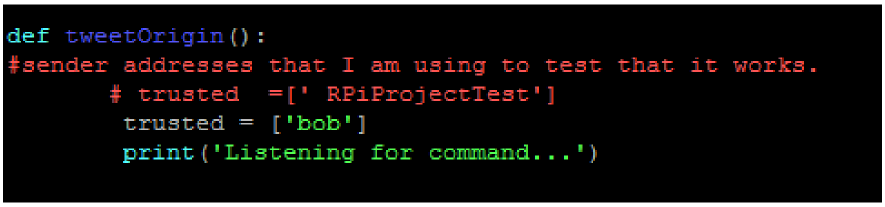
If new tweet by @RPiProjectTest with hashtag #testing, then make a web request

You tweeted: testing the project again #testing – view more

Show details

Applet ran
Apr 27 - 2:50 PM

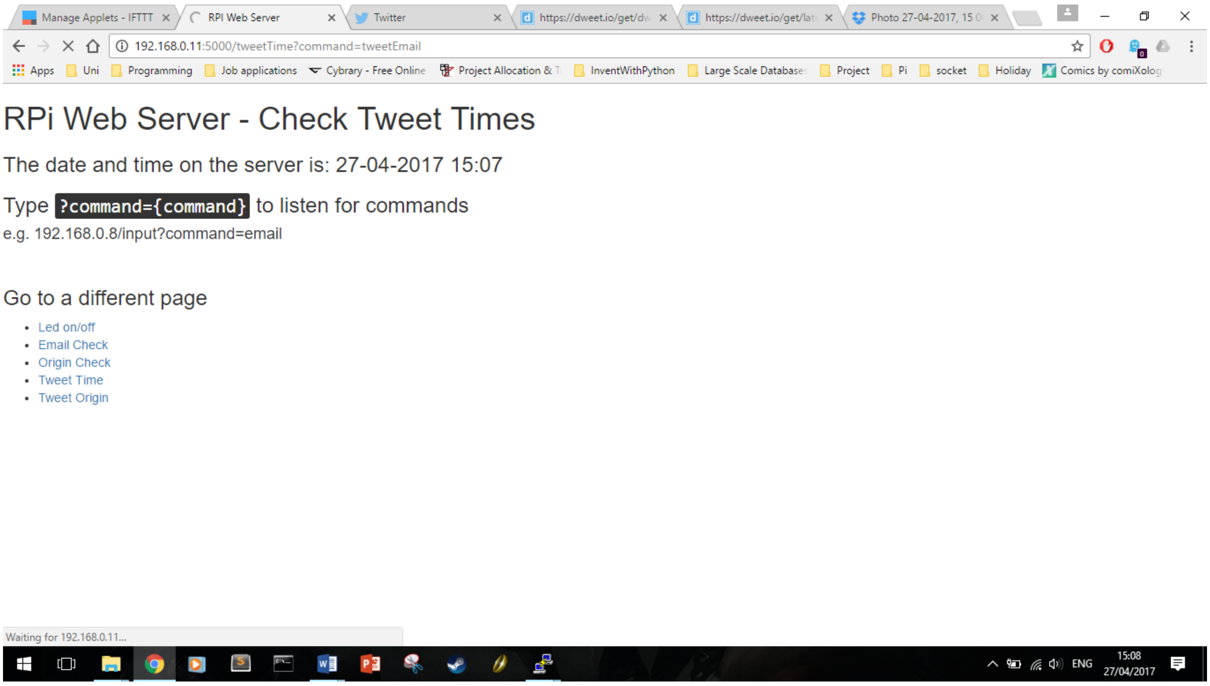
If maker Event "originReceived", then send a notification from the IFTTT app



```
def tweetOrigin():
#sender addresses that I am using to test that it works.
       # trusted  =[' RPiProjectTest']
        trusted = ['bob']
        print('Listening for command...')
```

Configuration File

10.2.7 Test number 7



192.168.0.11:5000/tweetTime?command=tweetEmail

RPi Web Server - Check Tweet Times

The date and time on the server is: 27-04-2017 15:07

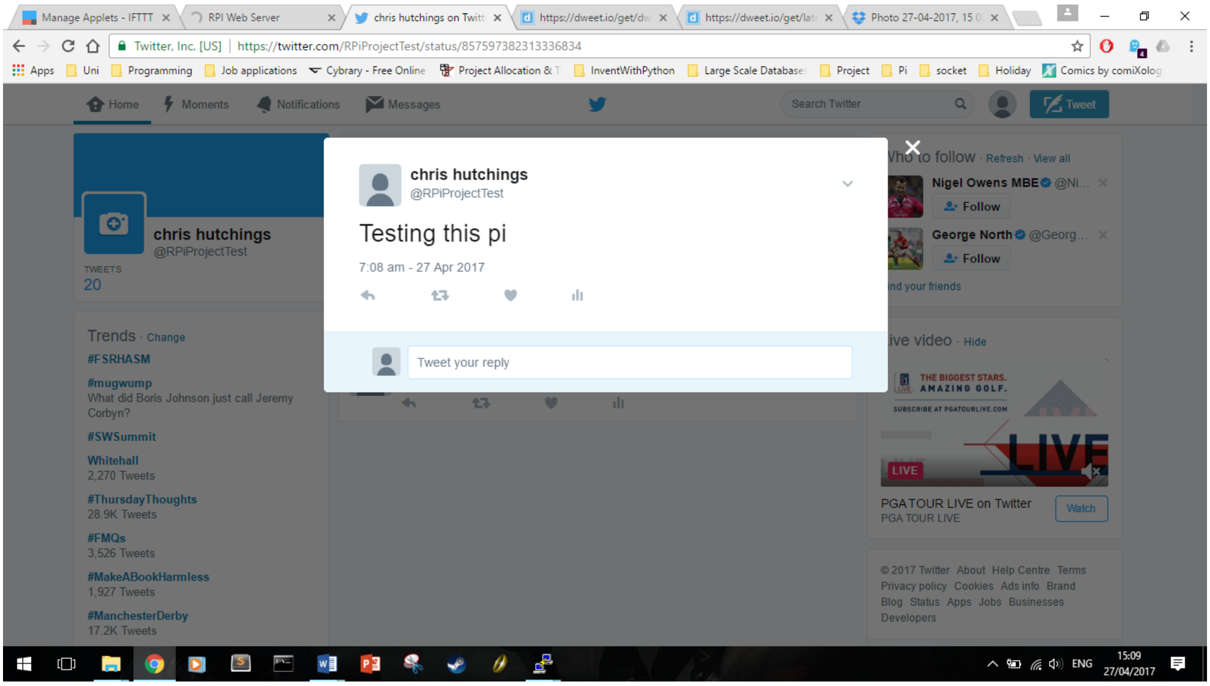
Type `?command={command}` to listen for commands

e.g. 192.168.0.8/input?command=email

Go to a different page

- Led on/off
- Email Check
- Origin Check
- Tweet Time
- Tweet Origin

Waiting for 192.168.0.11...



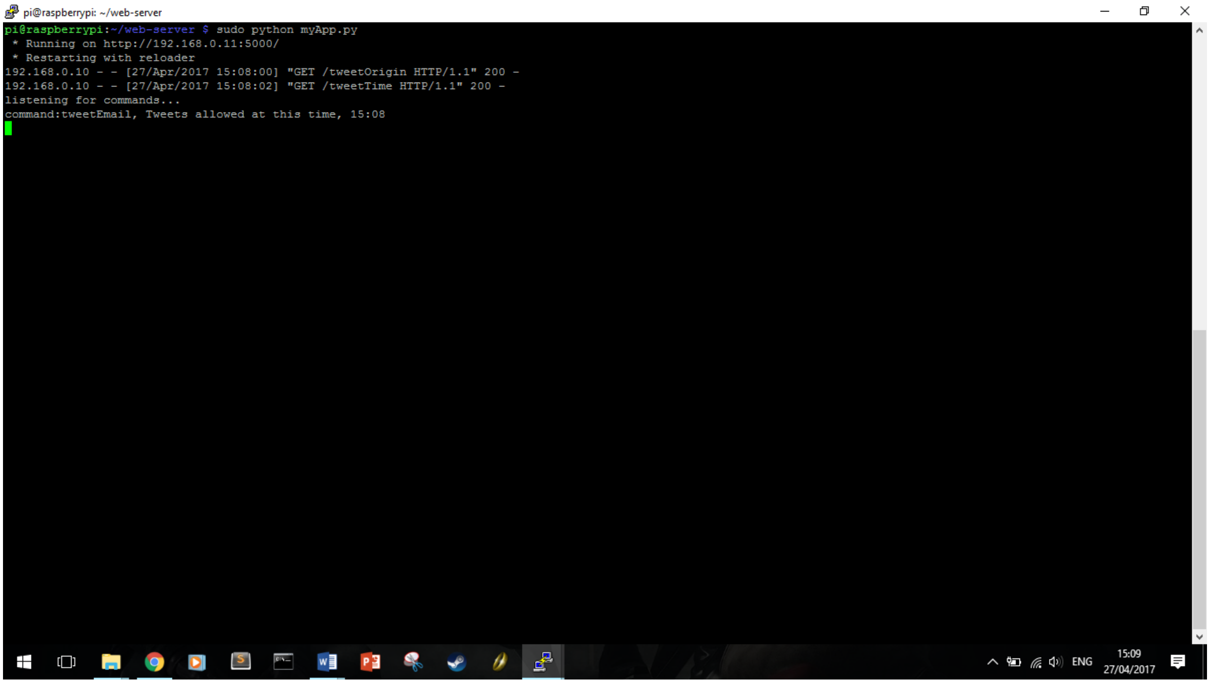
Twitter, Inc. [US] | https://twitter.com/RPiProjectTest/status/857597382313336834

chris hutchings
@RPiProjectTest

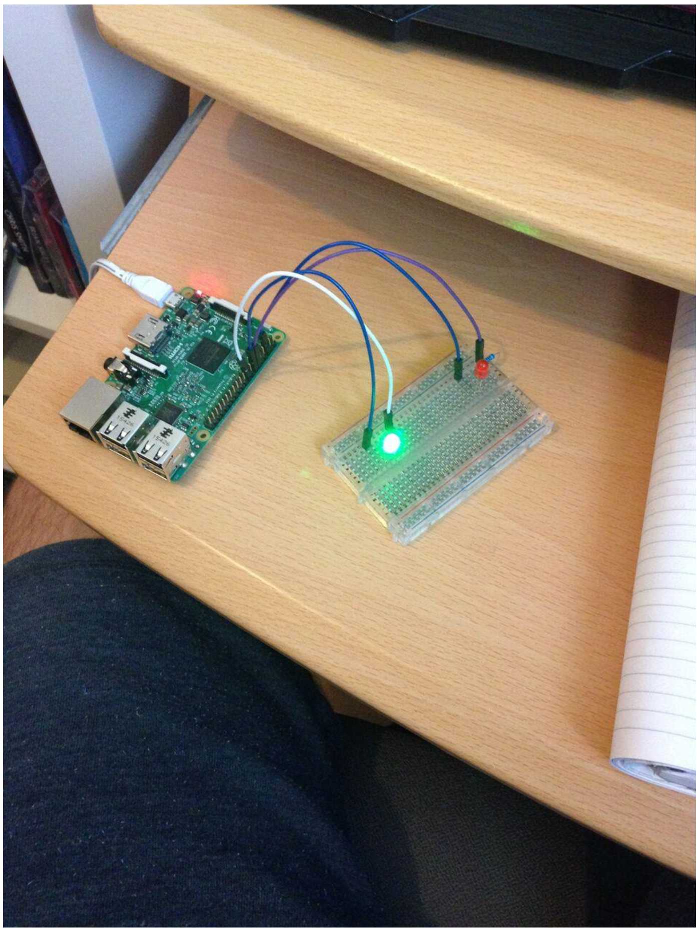
Testing this pi

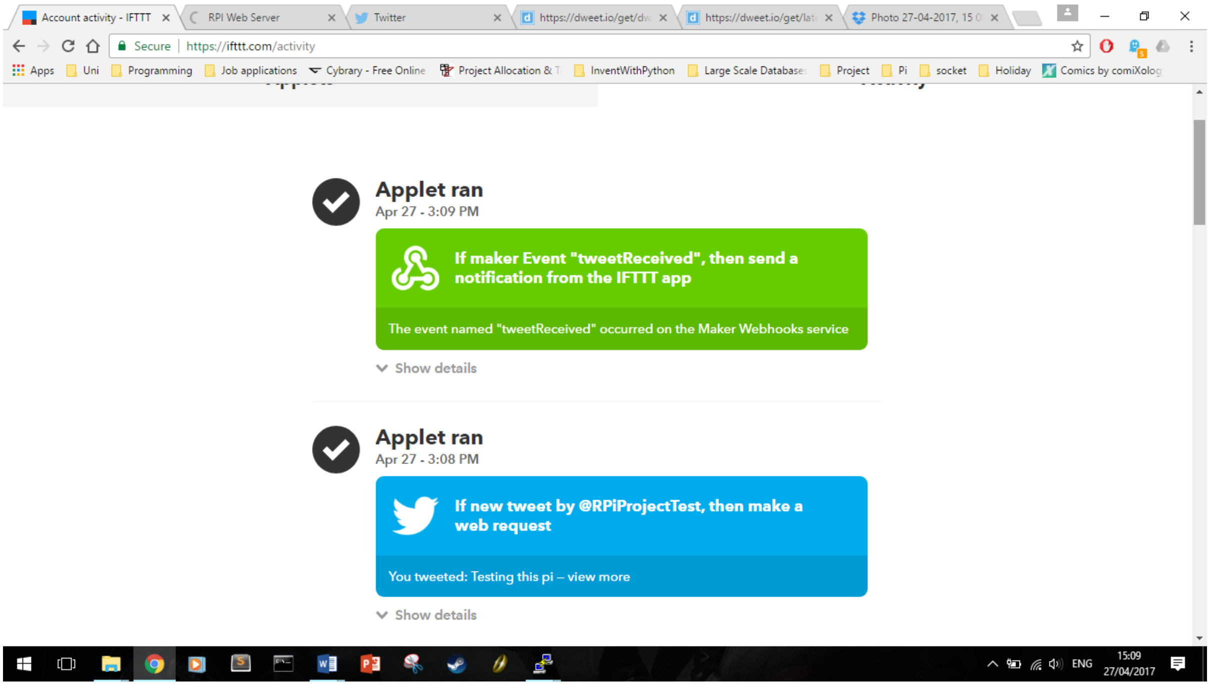
7:08 am - 27 Apr 2017

Tweet your reply



```
pi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
192.168.0.10 - - [27/Apr/2017 15:08:00] "GET /tweetOrigin HTTP/1.1" 200 -
192.168.0.10 - - [27/Apr/2017 15:08:02] "GET /tweetTime HTTP/1.1" 200 -
listening for commands...
command:tweetEmail, Tweets allowed at this time, 15:08
```





Applet ran
Apr 27 - 3:09 PM

If maker Event "tweetReceived", then send a notification from the IFTTT app

The event named "tweetReceived" occurred on the Maker Webhooks service

Show details

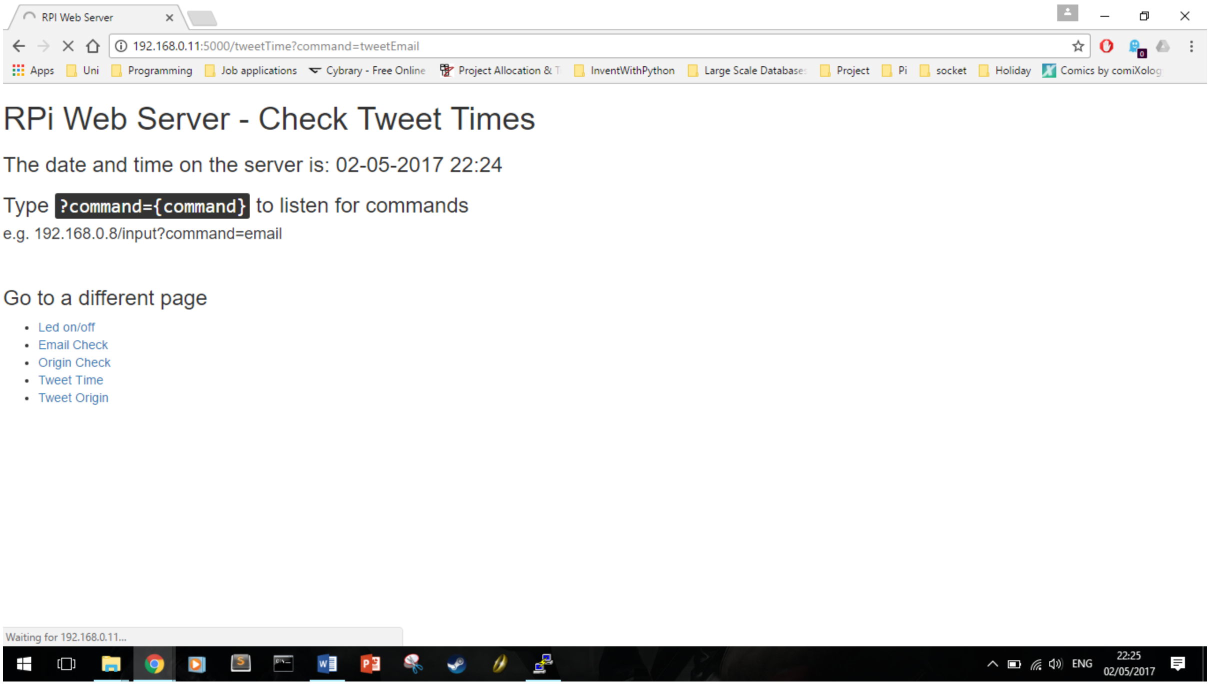
Applet ran
Apr 27 - 3:08 PM

If new tweet by @RPiProjectTest, then make a web request

You tweeted: Testing this pi – view more

Show details

10.2.8 Test number 8



RPi Web Server - Check Tweet Times

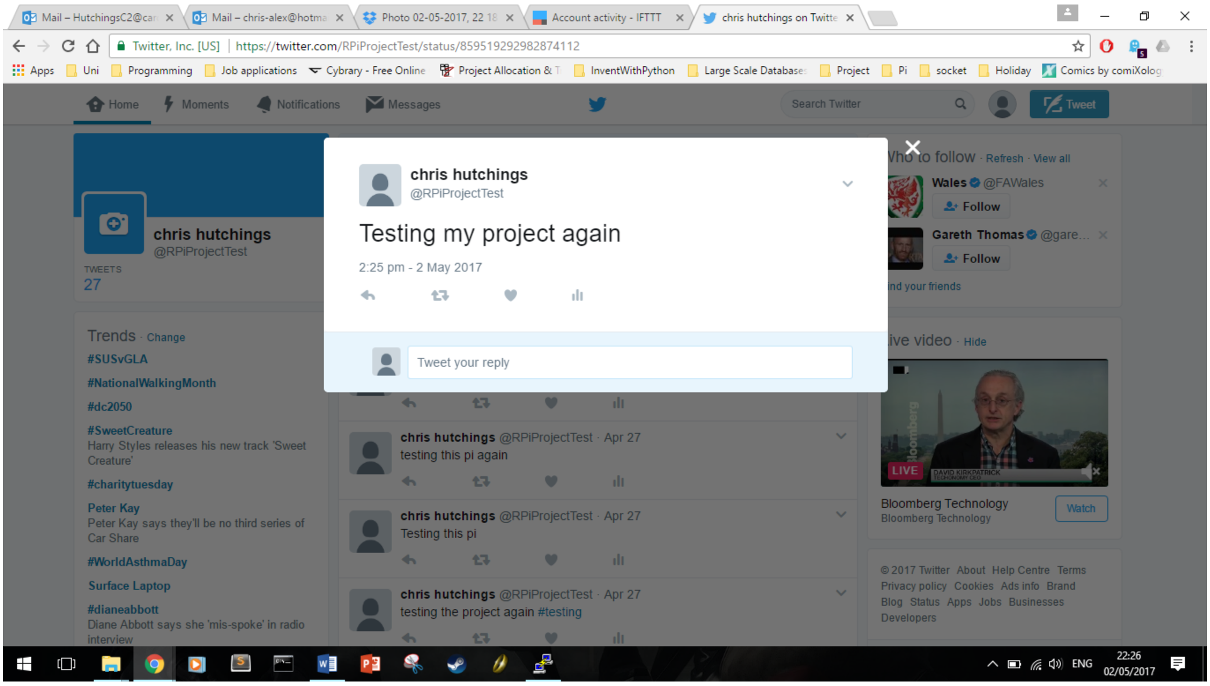
The date and time on the server is: 02-05-2017 22:24

Type `?command={command}` to listen for commands

e.g. 192.168.0.8/input?command=email

Go to a different page

- Led on/off
- Email Check
- Origin Check
- Tweet Time
- Tweet Origin



https://twitter.com/RPiProjectTest/status/859519292982874112

chris hutchings
@RPiProjectTest

Testing my project again

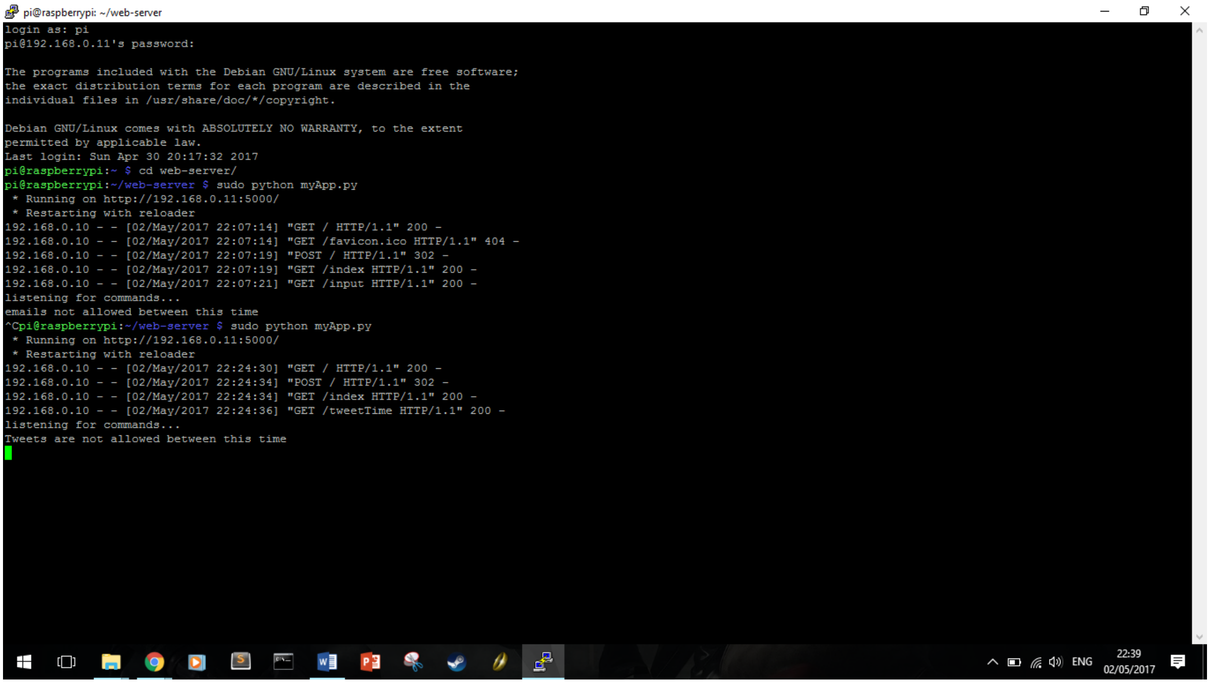
2:25 pm - 2 May 2017

Tweet your reply

chris hutchings @RPiProjectTest · Apr 27
testing this pi again

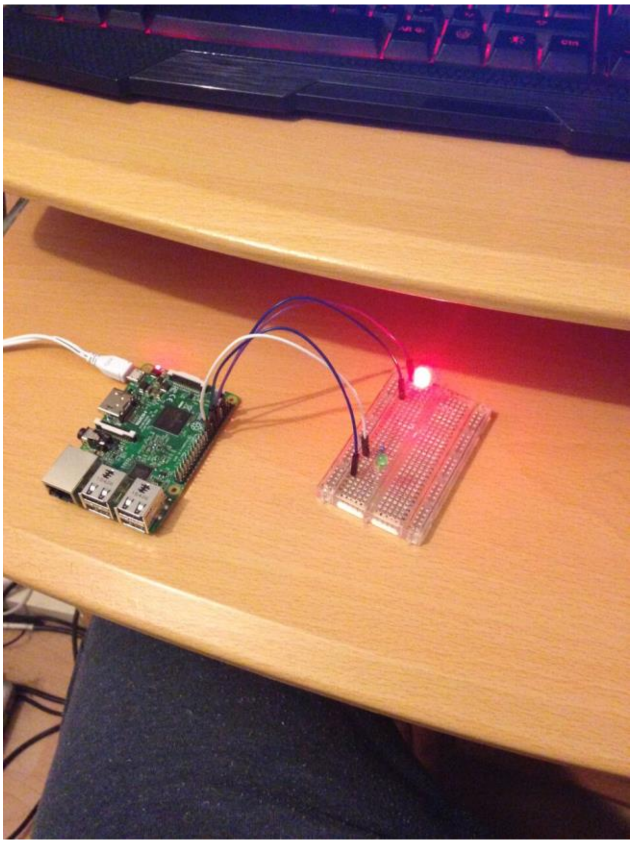
chris hutchings @RPiProjectTest · Apr 27
Testing this pi

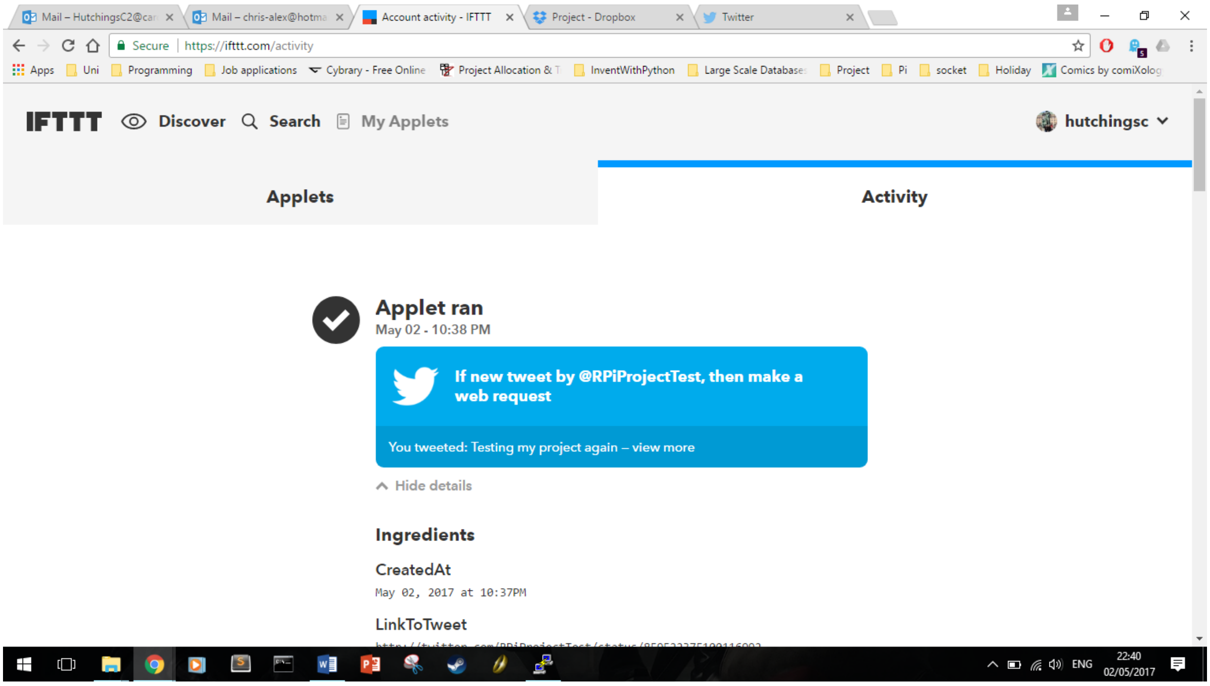
chris hutchings @RPiProjectTest · Apr 27
testing the project again #testing



pi@raspberrypi: ~/web-server

```
login as: pi
pi@192.168.0.11's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Apr 30 20:17:32 2017
pi@raspberrypi:~ $ cd web-server/
pi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
192.168.0.10 - - [02/May/2017 22:07:14] "GET / HTTP/1.1" 200 -
192.168.0.10 - - [02/May/2017 22:07:14] "GET /favicon.ico HTTP/1.1" 404 -
192.168.0.10 - - [02/May/2017 22:07:19] "POST / HTTP/1.1" 302 -
192.168.0.10 - - [02/May/2017 22:07:19] "GET /index HTTP/1.1" 200 -
192.168.0.10 - - [02/May/2017 22:07:21] "GET /input HTTP/1.1" 200 -
listening for commands...
emails not allowed between this time
^Cpi@raspberrypi:~/web-server $ sudo python myApp.py
 * Running on http://192.168.0.11:5000/
 * Restarting with reloader
192.168.0.10 - - [02/May/2017 22:24:30] "GET / HTTP/1.1" 200 -
192.168.0.10 - - [02/May/2017 22:24:34] "POST / HTTP/1.1" 302 -
192.168.0.10 - - [02/May/2017 22:24:34] "GET /index HTTP/1.1" 200 -
192.168.0.10 - - [02/May/2017 22:24:36] "GET /tweetTime HTTP/1.1" 200 -
listening for commands...
Tweets are not allowed between this time
```





Applets

Activity

Applet ran

May 02 - 10:38 PM

If new tweet by @RPiProjectTest, then make a web request

You tweeted: Testing my project again – view more

Hide details

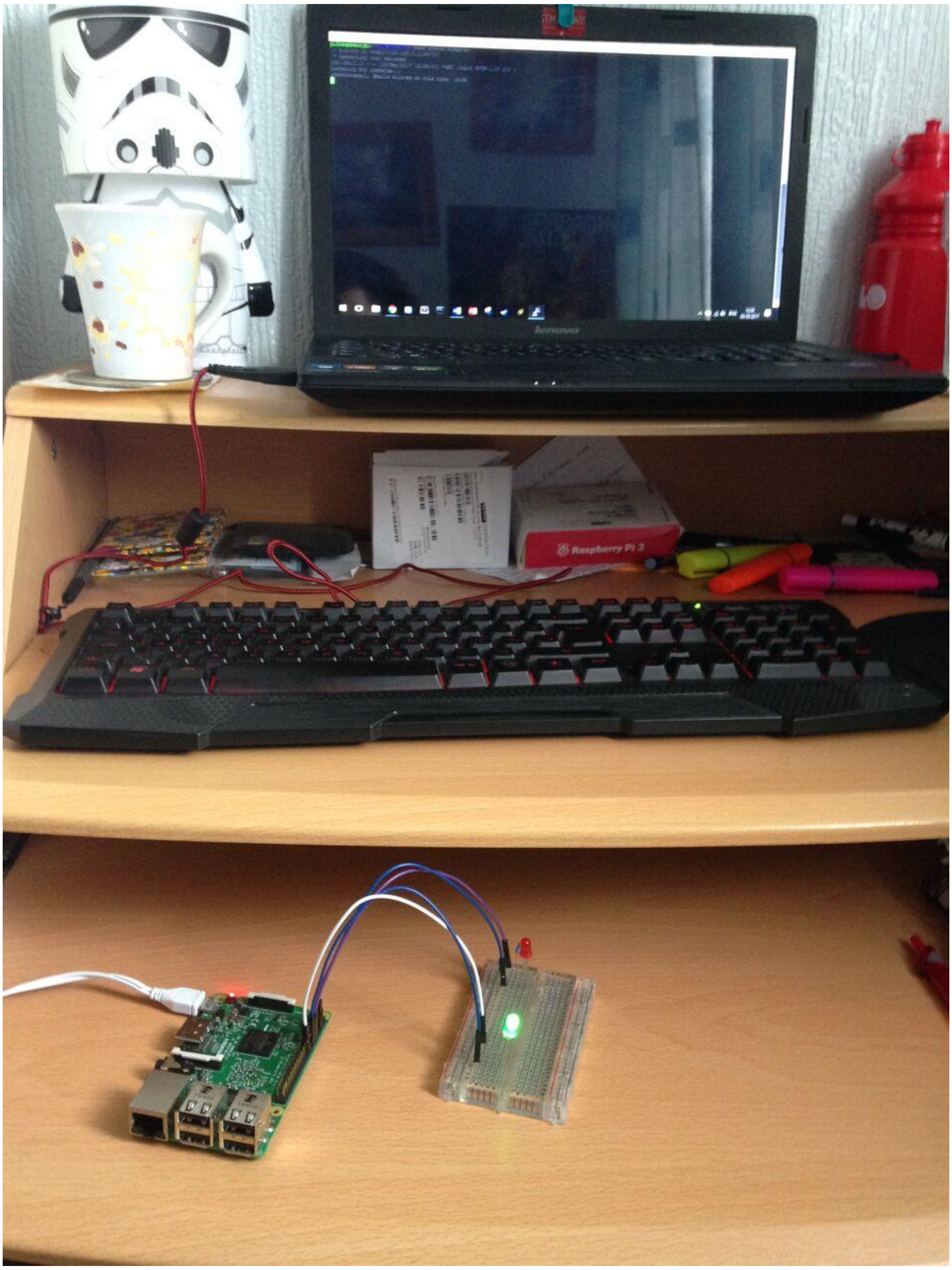
Ingredients

CreatedAt

May 02, 2017 at 10:37PM

LinkToTweet

http://twitter.com/RPiProjectTest/status/



Setup of the solution, laptop running putty to control the Raspberry Pi, the Raspberry Pi connected and running. The Pi LED is green due to a successful criteria being met for the policy.