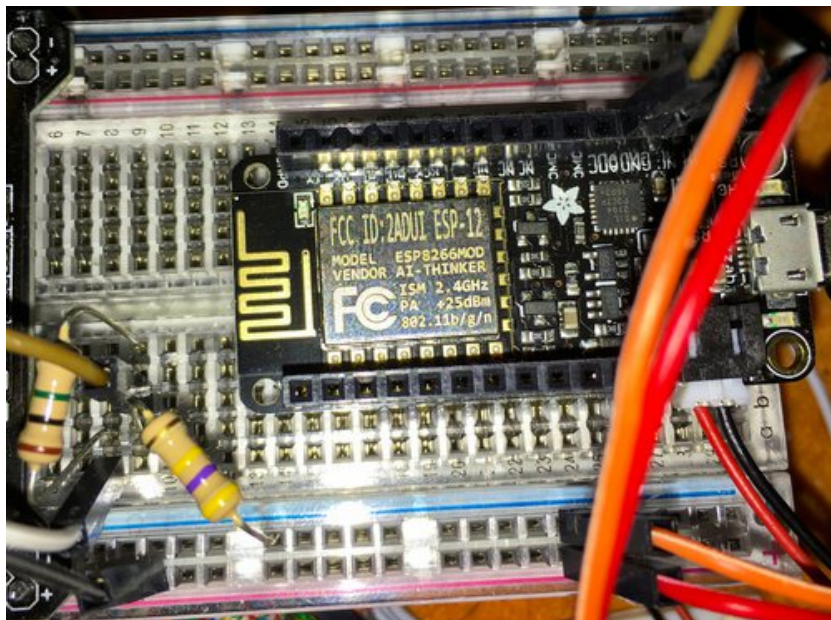


□

## DIY ESP8266 Home Security with Lua and MQTT

Created by Philip Moyer



Last updated on 2016-08-18 01:01:05 PM UTC

# Guide Contents

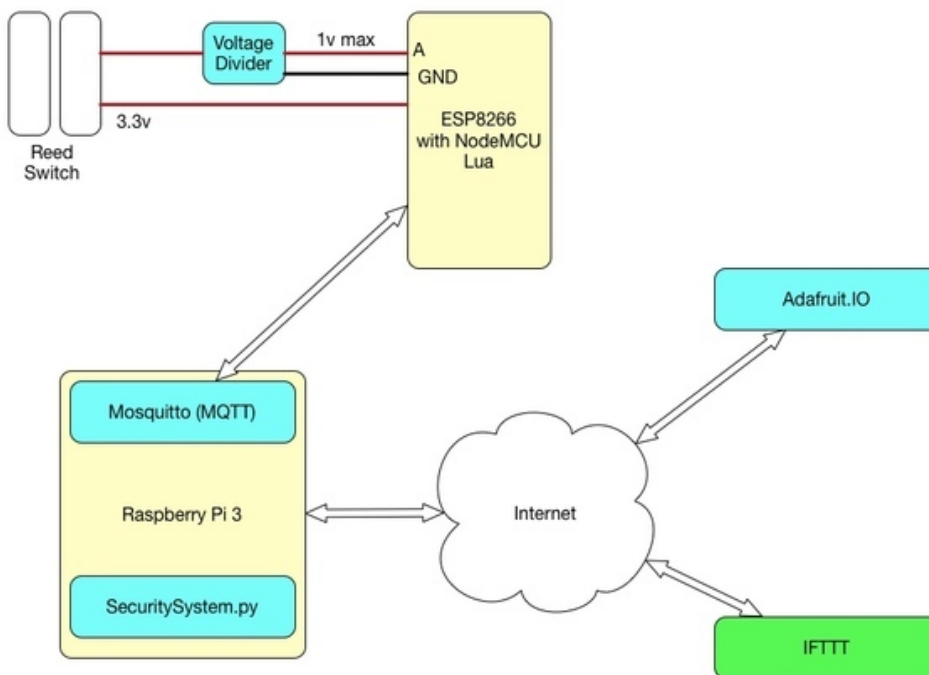
Guide Contents	2
Overview	3
What You'll Need	6
Building the Hardware	7
Programming the ESP8266 With Lua	10
Step 1 - Download the Software	10
Step 2 - Load Flash Memory on the ESP8266	13
Step 3 - Reset the ESP8266	13
Configuring MQTT on the Raspberry Pi	15
Step 1 - Install the Mosquitto (MQTT) Components	15
Step 2 - Configure Mosquitto and Restart the Service	15
Step 3 - Copy in the security.py Program and Edit it for Your Installation	16
Step 4 - Configure security.py to Run at Boot	19
Step 5 - Start security.py	20
Configuring io.adafruit.com	21
Step 1 - Create a feed for your alarm data	21
Step 2 - Create the dashboard for the alarms	22
Troubleshooting	27
Configuring IFTTT to Text	28
Step 1 - Add the Adafruit.IO monitor	29
Alternative Designs and Enhancements	34
How to Re-flash Your ESP8266	36
Step 1 - Getting the firmware	36
Step 2 - Getting the tool to flash an ESP8266	38
Step 3 - Flash the ESP8266	38

# Overview

One of the things I've noticed about home security systems is how expensive they are! The monetization model for home security is to not worry about making money on the hardware but sell the monitoring service at high rates. This is needed to support call centers for monitoring all the devices on a 24x7 basis. I'd love to have an alarm system, but they're prohibitively expensive. We Makers can certainly do better!

The goal of the system is to detect doors or windows opening. If you don't expect the door or window to open, you know something is up. This tutorial only talks about the door/window sensors and the alarm sequence. There are all kinds of things you can add to this alarm system once you have it constructed: smoke/fire sensors, carbon monoxide detectors, and security cameras, to name a few.

Here's the design for the system:



Now, here's how it works. The reed switch (Adafruit product ID 375) is a normally open switch. This means that when the magnet is not close to the wired switch there's no connection between the two wires. When the magnet is close to the wired switch the switch closes and current can flow.

There is one analog pin on the ESP8266, whether it's the Feather, the breakout, or some other ESP8266-based device. The input voltage on that pin is one volt, maximum. Since the Feather provides 3.3v, we need a voltage divider to limit the input voltage to the 0-1 volt range.

The analog pin, which is connected to an analog-to-digital converter, will see a value of one volt when the door or window is closed. This means it reads 1024. When the door or window is opened, the voltage will drop to something "close to" zero, modulo the natural float fluctuation in the line. This means the digital reading on the

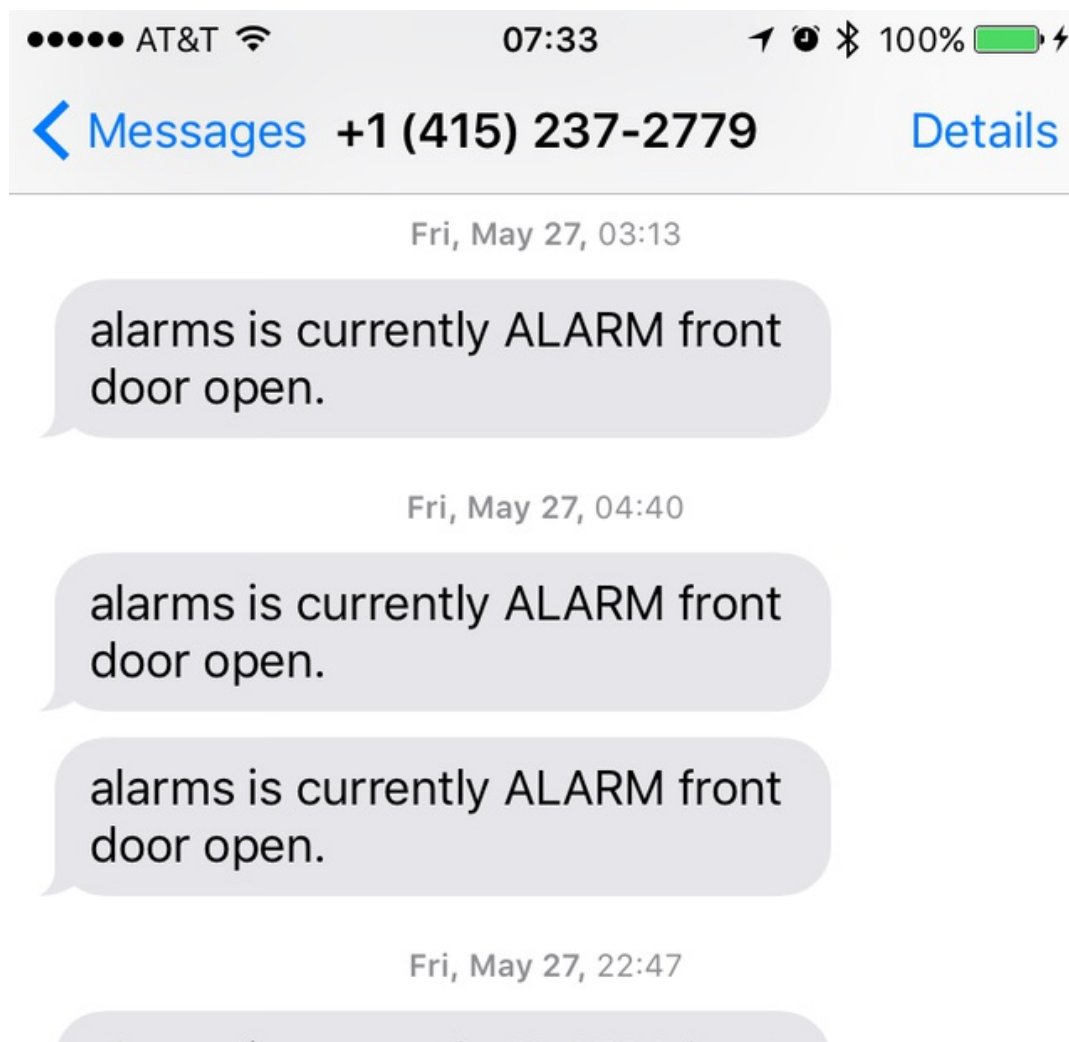
line will be something less than 50.

There's a timer set on the ESP8266 that wakes up every second and reads the analog pin. It's already connected to the wireless network (the file *init.lua* does that for us - more on that later) so we don't have to worry about managing the connection. It opens a connection to a local MQTT broker running on a Raspberry Pi 3 that's also on the local network. It inserts the sensor ID and the analog pin reading into the MQTT topic queue.

Then there is a Python program running on the same Raspberry Pi. It subscribes to the alarms topic on the MQTT broker and reads the sensor IDs and ADC values. This is the program that actually keeps track of the current state of each sensor in order to detect state changes. When the door or window goes from open to closed, it generates an "INFO" event. When the door or window goes from closed to open, however, the program generates an "ALARM" message. Where does that message go? Why, into an Adafruit.IO text feed, of course! This gets the alarm message out of the local network and into the cloud where it can be processed by other systems.

Finally, there's an If This Then That (IFTTT) recipe that looks for ALARM events in the Adafruit.IO topic queue. When it sees one, it texts the alarm message to the pre-configured phone number (which happens to be my cell phone).

Putting it all together, when a door or window opens I get a text on my phone and iWatch that tells me which door/window has opened. If it's not expected, I can take appropriate action.



alarms is currently ALARM front door open.

alarms is currently ALARM front door unknown.

alarms is currently ALARM front door open.

Saturday 09:46

alarms is currently ALARM front door open.



Text Message

Send

Next: What You'll Need

# What You'll Need

Here's the bill of materials (BoM) of all the parts you'll need to complete this project. You can source the parts through Adafruit or, if you need to, through various online wholesalers like DigiKey. Obviously, I recommend you buy your bits through Adafruit like I do.

1. [Magnetic contact switch](http://adafru.it/375) (<http://adafru.it/375>) (Adafruit Product ID 375). You will need one of these for each door and window you want to protect.
2. Adafruit [Feather HUZZAH ESP8266](http://adafru.it/2821) (<http://adafru.it/2821>) (Adafruit Product ID 2821) or Adafruit [HUZZAH ESP8266 breakout](http://adafru.it/2471) (<http://adafru.it/2471>) (Adafruit Product ID 2471). You will need one of these for each door and window you want to protect.
3. Adafruit [Lithium Ion Polymer \(LiPo\) battery](http://adafru.it/328) (<http://adafru.it/328>), 3.7v in whatever size you like. I use the 2500 mAh version (Adafruit Product ID 328). You will need one of these for each door and window you want to protect.
4. [Five-volt switching power supply](http://adafru.it/1995) (<http://adafru.it/1995>) with MicroUSB connector (Adafruit Product ID 1995) to power the Feathers. You will need one of these for each door and window you want to protect, as well as one for the Raspberry Pi 3 "control center."
5. [Resistors](http://adafru.it/oxa). (<http://adafru.it/oxa>) The voltage divider will use a 1M Ohm and a 470K Ohm resistor. You will need a voltage divider for each door and window you want to protect.
6. [Raspberry Pi 3](http://adafru.it/3055). (<http://adafru.it/3055>) (Adafruit Product ID 3055) to run the MQTT broker and the Python security system program.
7. [A class-10 SDHC micro card](http://adafru.it/2692) (<http://adafru.it/2692>) for the Raspberry Pi operating system (OS).
8. [An account on Adafruit.IO](http://adafru.it/oxb) (<http://adafru.it/oxb>)
9. An account on [If This Then That \(IFTTT\)](http://adafru.it/oxc). (<http://adafru.it/oxc>)
10. A mobile phone that can receive SMS text messages.

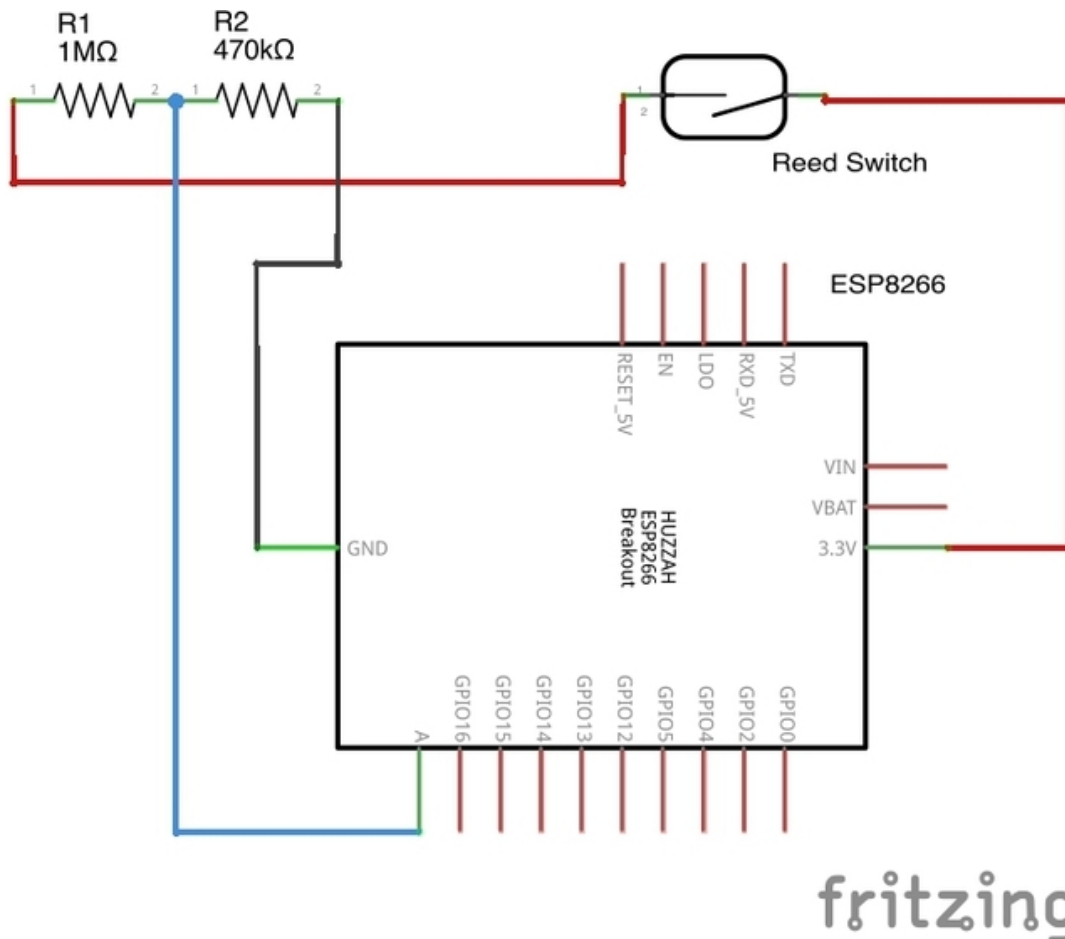
Finally, I recommend a high capacity deep cycle UPS system so your cable/fiber modem, wireless access point, and Raspberry Pi will keep running even when mains power is not available.

**Next:** Building the Hardware!

# Building the Hardware

The first thing you'll need to do is build all the sensors and prepare the Raspberry Pi 3. The sensors consist of a reed switch, a voltage divider, a Feather HUZZAH ESP8266 or ESP8266 breakout, a LiPo backup battery, a power supply, and a half-size breadboard (you can solder all the wiring together if you prefer).

First, we need to build the voltage divider out of 1M Ohm and 470K Ohm resistors. 3.3v is connected from the ESP8266 to one side of the door/window open reed switch. The other side of the switch is wired to the voltage divider (more on the voltage divider below). Here is a schematic of the wiring:

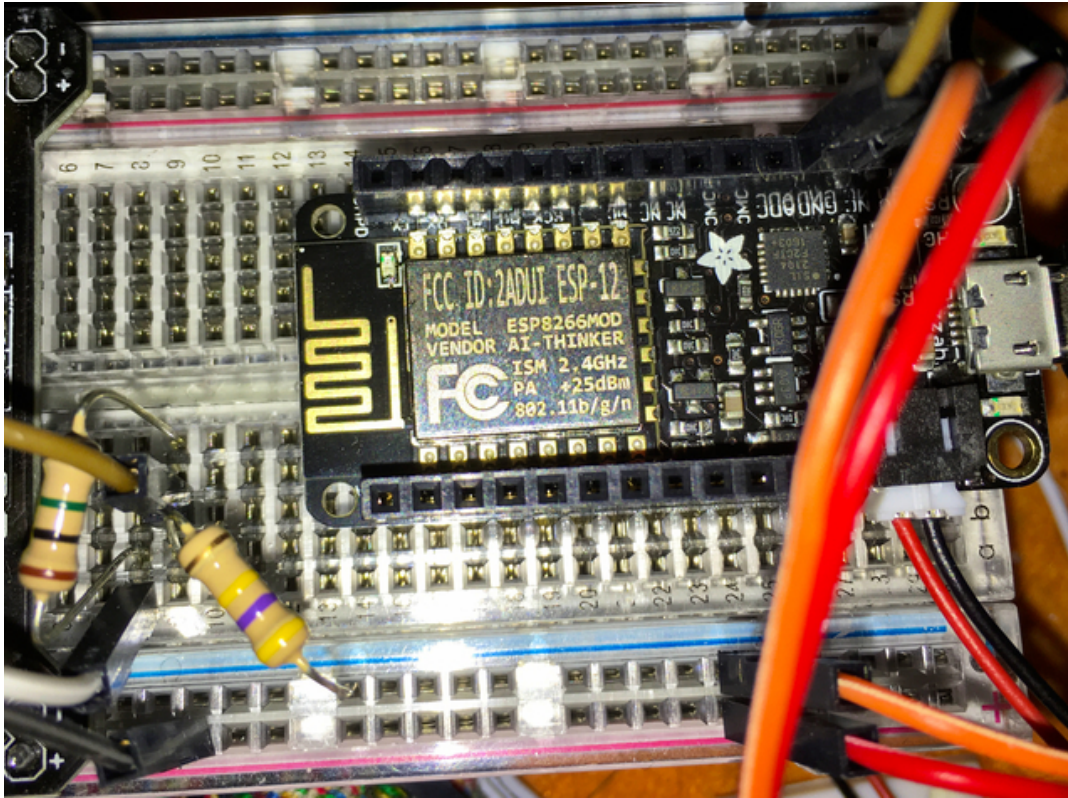


The voltage divider reduces the 3.3v range coming out of the reed switch to the 0-1v range needed by the analog pin on the ESP8266.

**This schematic uses the Analog pin to sense whether the reed switch is open or closed, you can also use any digital I/O pin. If you're sensing a light or photocell temperature sensor, you'll want to stick with analog input. Adjust your design as necessary!**

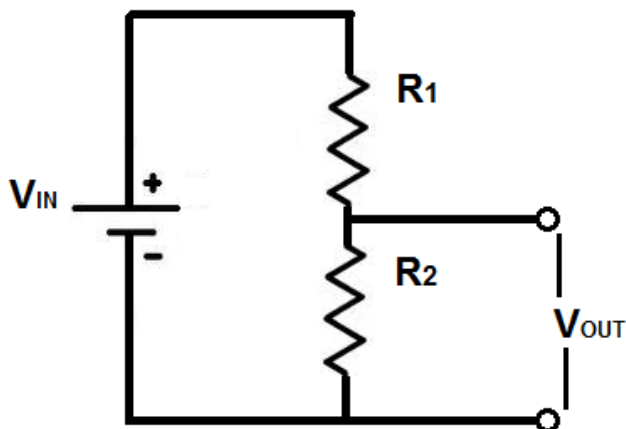
Now, I used [half-sized solderless breadboards](http://adafru.it/64) (<http://adafru.it/64>) (Adafruit Product ID 64) for building my alarm system sensors. As an alternative, you can solder the voltage divider and reed switch directly into the circuit using your preferred low-voltage wiring (I'd recommend [22 AWG hook-up wire](http://adafru.it/1311) (<http://adafru.it/1311>) - Adafruit Product ID 1311). This is particularly easy if you don't solder headers into the ESP8266 board you're using.





Just follow the schematic to build the sensors. Remember, you'll need one sensor for each door and window you want to protect. We'll modify the Lua code going onto each sensor to configure a unique sensor ID for each device. The *SecuritySystem.py* Python program, which runs on the Raspberry Pi, has a translation table from sensor IDs to human-meaningful text, e.g., sensor ID "sensor001" = "front door." This allows the program to insert a useful message into the Adafruit.IO topic queue (more about that in a bit).

These are really very simple circuits to build. The most complicated part is the voltage divider. There is a useful [WikiPedia article on voltage dividers](http://adafru.it/osc) (<http://adafru.it/osc>), and there are many voltage divider calculators on the Internet. The one I used to design the sensor's voltage divider is from [Learning About Electronics](http://adafru.it/osd) (<http://adafru.it/osd>).



Of course, if you know the formula for the calculation, you can just use that directly. I'm not a EE, though, so I'd have had to look it up anyway. Incidentally, the voltage divider is the first "project" in Hayes and Horowitz's classic "[Learning the Art of Electronics](http://adafru.it/3066)" (<http://adafru.it/3066>) (Adafruit Product ID 3066) - it starts on page 11.



Test the voltage divider by applying 3.3v to Vin and measuring the voltage between the two resistors with your meter. It should read about one volt. Once that's working correctly, connect 3.3v to one side of the reed switch and connect the other side to Vin of the voltage divider. Connect one side of R2 to R1 and the other to ground. Finally, connect Vout (the connection between the two resistors) to the single Analog pin on the ESP8266. The voltage divider assures that we will not exceed the ESP8266's input voltage restriction on the analog pin.

You will also need to set up or have available a Raspberry Pi 2 or 3 (I use a 3) to run the MQTT broker and the *SecuritySystem.py* Python program. Follow the usual process for configuring Jessie on the Raspberry Pi and connect it to your network. Wired or wireless will do, though for bandwidth reasons I used a wired connection.

That's it! It's really quite simple.

Next: Programming the ESP8266 with Lua

# Programming the ESP8266 With Lua

The ESP8266 is a *wonderful* little ecosystem and I've become quite fond of it lately. The Feather HUZZAH ESP8266 comes pre-flashed with the [NodeMCU](http://adafru.it/ose) (<http://adafru.it/ose>) Lua interpreter, as does the HUZZAH ESP8266 breakout. This means you can program the WiFi chip directly using Lua.

When you are programming the ESP8266 with Lua, you can write any Lua programs you like to the board's flash memory. If you create a program named *init.lua* it will run when the board resets.

If you have an infinite loop in *init.lua* you WILL brick your device!!! The only solution is to re-flash the firmware.

I use an *init.lua* that I found by a search on the Internet while I was trying to learn enough about Lua and the ESP8266 to avoid bricking my device (and I still did it once). If you have an error in *init.lua*, one of two things will happen. First, it could fall through to the Lua interpreter. Second, it could go into an infinite loop that you cannot interrupt, bricking the device. If (when?) this happens, see the page titled "How to Re-flash Your ESP8266."

## Step 1 - Download the Software

There are two programs for the ESP8266 that make up the security system. They are *init.lua* and *security.lua*. Here is *init.lua*:

```
SSID = "YOUR_WIFI_SSID"
APPWD = "YOUR_WIFI_PASSWORD"
CMDFILE = "ping.lua" -- File that is executed after connection

wifiTrys = 15 -- Counter of trys to connect to wifi
NUMWIFITRYS = 200 -- Maximum number of WIFI Testings while waiting for connection

function launch()
  print("Connected to WIFI!")
  print("IP Address: " .. wifi.sta.getip())
  -- Call our command file. Note: if you foul this up you'll brick the device!
  dofile("security.lua")
  makeConn()
end

function checkWIFI()
  if ( wifiTrys > NUMWIFITRYS ) then
    print("Sorry. Not able to connect")
  else
    ipAddr = wifi.sta.getip()
    if ( ( ipAddr ~= nil ) and ( ipAddr ~= "0.0.0.0" ) ) then
      tmr.alarm( 1 , 500 , 0 , launch )
    else
      -- Reset alarm again
      tmr.alarm( 0 , 2500 , 0 , checkWIFI)
      print("Checking WIFI..." .. wifiTrys)
      wifiTrys = wifiTrys + 1
    end
  end
end

print("-- Starting up! ")

-- Lets see if we are already connected by getting the IP
```

```

ipAddr = wifi.sta.getip()
if ( ( ipAddr == nil ) or ( ipAddr == "0.0.0.0" ) ) then
  -- We aren't connected, so let's connect
  print("Configuring WIFI...")
  wifi.setmode( wifi.STATION )
  wifi.sta.config( SSID , APPWD )
  print("Waiting for connection")
  tmr.alarm( 0 , 2500 , 0 , checkWIFI )
else
  -- We are connected, so just run the launch code.
  launch()
end

```

Just download the above program to your laptop for now. Don't upload it to your ESP8266 yet!

Here is the program *security.lua*:

```

-- #####
-- security.lua - functions to support reed switch magnetic door and
-- window normally-open, which will be part of a security system.
--
-- Note: these are just dumb sensors programmed to insert their
-- values into the local network MQTT queue (aka topic). The
-- Raspberry Pi (either the one running the broker or another one)
-- has all the smarts about what to do with the data in the topic
-- queue.
--
-- Note 2: I'll be improving this code as I learn more about Lua.
-- I recommend the book "Programming in Lua" by Roberto
-- Ierusalimsky (one of the designers of Lua).
--
-- Phil Moyer
-- Adafruit
--
-- May 2016
--
-- This code is open source, released under the BSD license. All
-- redistribution must include this header.
-- #####

-- #####
-- Global variables and parameters.
-- #####

sensorID = "security_001" -- a sensor identifier for this device
tgtHost = "MQTT_BROKER_IP_ADDR" -- target host (broker)
tgtPort = 1883 -- target port (broker listening on)
mqttUserID = "MQTT_USER_ID" -- account to use to log into the broker
mqttPass = "MQTT_USER_PASSWORD" -- broker account password
mqttTimeOut = 120 -- connection timeout
dataInt = 1 -- data transmission interval in seconds
topicQueue = "/security" -- the MQTT topic queue to use

-- You shouldn't need to change anything below this line. -Phil --

-- #####
-- Functions
-- #####

-- Function pubEvent() publishes the sensor value to the defined queue.

function pubEvent()

```

```

rv = adc.read(0) -- read light sensor
pubValue = sensorID .. " " .. rv -- build buffer
print("Publishing to " .. topicQueue .. ": " .. pubValue) -- print a status message
mqttBroker:publish(topicQueue, pubValue, 0, 0) -- publish
end

```

```

-- Reconnect to MQTT when we receive an "offline" message.

```

```

function reconn()
  print("Disconnected, reconnecting....")
  conn()
end

```

```

-- Establish a connection to the MQTT broker with the configured parameters.

```

```

function conn()
  print("Making connection to MQTT broker")
  mqttBroker:connect(tgtHost, tgtPort, 0, function(client) print ("connected") end, function(client, reason) print("failed reason: "..reason) end)
end

```

```

-- Call this first! --
-- makeConn() instantiates the MQTT control object, sets up callbacks,
-- connects to the broker, and then uses the timer to send sensor data.
-- This is the "main" function in this library. This should be called
-- from init.lua (which runs on the ESP8266 at boot), but only after
-- it's been vigorously debugged.
--
-- Note: once you call this from init.lua the only way to change the
-- program on your ESP8266 will be to reflash the NodeCMU firmware!

```

```

function makeConn()
  -- Instantiate a global MQTT client object
  print("Instantiating mqttBroker")
  mqttBroker = mqtt.Client(sensorID, mqttTimeout, mqttUserID, mqttPass, 1)

```

```

  -- Set up the event callbacks
  print("Setting up callbacks")
  mqttBroker:on("connect", function(client) print ("connected") end)
  mqttBroker:on("offline", reconn)

```

```

  -- Connect to the Broker
  conn()

```

```

  -- Use the watchdog to call our sensor publication routine
  -- every dataInt seconds to send the sensor data to the
  -- appropriate topic in MQTT.
  tmr.alarm(0, (dataInt * 1000), 1, pubEvent)
end

```

```

-- #####
-- "Main"
-- #####

```

```

-- No content. -prm

```

You will need to change the parameters appropriately for your network and MQTT configuration. Also, mark your sensors somehow with the sensorID value. Make sure you change this in *security.lua* for every device you make. The sensorID needs to be unique or the alarm system will generate alarms with the wrong ingress point

listed! You should mark the sensorID on the sensor somehow (masking tape and sharpie, zip-top bag and a sticky note, toe tag, etc.) so you don't get confused when you're installing them around the house or office.

I use sequential numbers for the sensor ID, as you can see from *thesecurity.lua* code.

At this point you could re-write the code to keep track of state changes in the sensor and only send an event to the MQTT broker when the state changes. I'm only beginning to learn Lua, so that was a little beyond my skills, so I put all the smarts of the system into the Python monitor program. Speaking of learning Lua, I recommend Roberto Ierusalimsky's book "[Programming in Lua \(http://adafru.it/osf\)](http://adafru.it/osf)," which is a great introduction to Lua by one of Lua's creators. Learning Lua will, I think, significantly enhance your enjoyment of the ESP8266 devices.

## Step 2 - Load Flash Memory on the ESP8266

Once you have the code for the two Lua programs saved on your local machine and edited accordingly, you need to upload them to the flash memory of the ESP8266 device. I am on a Mac and use a tool called [luatool \(http://adafru.it/osA\)](http://adafru.it/osA) that just uploads programs into the ESP8266 flash.

Luatool is written in Python so it should run on any OS that supports Python. You can download luatool from its [GitHub repository \(http://adafru.it/osA\)](http://adafru.it/osA).

Here's what it looks like to run the tool:

```
Puller:luatool prms$ python ./luatool.py -p /dev/tty.SLAB_USBtoUART -f ~/ESP8266/blink.lua

->file.open("blink.lua", "w") -> ok
->file.close() -> ok
->file.remove("blink.lua") -> ok
->file.open("blink.lua", "w+") -> ok
->file.writeline([[-- Config]]) -> ok
->file.writeline([[local pin = 4          --> GPIO2]]) -> ok
->file.writeline([[local value = gpio.LOW]]) -> ok
->file.writeline([[local duration = 1000   --> 1 second]]) -> ok
->file.writeline([[ ]]) -> ok
->file.writeline([[ ]]) -> ok
->file.writeline([[-- Function toggles LED state]]) -> ok
->file.writeline([[function toggleLED ( )]]) -> ok
->file.writeline([[if value == gpio.LOW then]]) -> ok
->file.writeline([[value = gpio.HIGH]]) -> ok
->file.writeline([[else]]) -> ok
->file.writeline([[value = gpio.LOW]]) -> ok
->file.writeline([[end]]) -> ok
->file.writeline([[ ]]]) -> ok
->file.writeline([[gpio.write(pin, value)]]) -> ok
->file.writeline([[end]]) -> ok
->file.writeline([[ ]]]) -> ok
->file.writeline([[ ]]]) -> ok
->file.writeline([[-- Initialise the pin]]) -> ok
->file.writeline([[gpio.mode(pin, gpio.OUTPUT)]]) -> ok
->file.writeline([[gpio.write(pin, value)]]) -> ok
->file.writeline([[ ]]]) -> ok
->file.writeline([[-- Create an interval]]) -> ok
->file.writeline([[tmr.alarm(0, duration, 1, toggleLED)]]) -> ok
->file.flush() -> ok
->file.close() -> ok
--->>> All done <<<---
Puller:luatool prms$
```

This is a sample program I wrote that just blinks the LED; it's not part of the security system. The example shows what it looks like to run luatool, though.

## Step 3 - Reset the ESP8266

When you have uploaded both programs, *security.lua* and *init.lua*, to flash, reset the ESP8266 with the reset button and your sensor should boot up and be working. You should connect to it with an appropriate serial device (FTDI friend, console cable, or USB cable depending on your ESP8266 board) and a terminal program like CoolTerm, then press the reset button. You should see the NodeMCU banner, followed by the WiFi connection, followed by some switch state information.

Change the `sensorID` parameter in *security.lua* for each sensor you flash and upload both Lua programs to each sensor. Once you have done this, you can install the switches on your doors and windows, then plug in the power supply for each sensor.

Next: Configuring MQTT on the Raspberry Pi



# Configuring MQTT on the Raspberry Pi

MQTT, which originally was an acronym for Message Queue Telemetry Transport, is a lightweight message queue protocol designed for small data packets sent across high latency, low bandwidth links. MQTT is a fairly simple protocol and it's perfect for Internet of Things projects. It's also perfect for this security system project!

The version of MQTT I use in this tutorial is called Mosquitto. It is available via *apt*, so installing it is quite easy. There are a number of steps in configuring the Raspberry Pi component of the security system. As I mentioned, I'm using a Raspberry Pi 3.



The steps you need to follow are, at a high level:

1. Install mosquitto (MQTT) components.
2. Configure mosquitto and restart the service.
3. Copy in the security.py program and edit it for your installation.
4. Configure security.py to run at boot.
5. Start security.py.

As I've mentioned earlier, I'm using the Raspberry Pi 3 running the latest version of Raspbian Jessie.

## Step 1 - Install the Mosquitto (MQTT) Components

Installing mosquitto is as easy as running a few apt commands. First, though, we make sure we're running the latest version of Jessie and that all the software is up to date. Here are the commands to run to upgrade the system and install the mosquitto components:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get dist-upgrade
sudo apt-get install mosquitto mosquitto-clients python-mosquitto
```

## Step 2 - Configure Mosquitto and Restart the Service

Mosquitto is controlled in two ways. First, the default configuration is in */etc/mosquitto/mosquitto.conf*. I recommend you not edit this file, however, and instead, use the second mechanism, which is a file with a .conf extension in */etc/mosquitto/conf.d*. I actually named mine *mosquitto.conf*, too, so the full path to the local configuration file is */etc/mosquitto/conf.d/mosquitto.conf*. This file is populated with example configurations by default, so you'll want to edit it for your local use. Here is the local configuration file I recommend:

```
# Config file for mosquitto
#
# See mosquitto.conf(5) for more information.
```

```
user mosquitto
max_queued_messages 200
message_size_limit 0
allow_zero_length_clientid true
allow_duplicate_messages false

listener 1883
autosave_interval 900
autosave_on_changes false
persistence true
persistence_file mosquitto.db
allow_anonymous true
password_file /etc/mosquitto/passwd
```

Once you have edited the configuration file, restart the service with the command

```
sudo systemctl restart mosquitto
```

## Step 3 - Copy in the security.py Program and Edit it for Your Installation

Here is the security.py program:

```
#####
# security.py - Monitors a Mosquitto MQTT queue for security events
# from an array of security sensors, detects critical changes in those
# sensor values, and injects alarms into an io.adafruit.com queue.
#
# Note: The hardware to do this is already developed (Feather Huzzah
# ESP8266 with NodeCMU), along with the Lua software to run on
# the ESP8266. The next development steps are:
# - write this python program
# - write the If This Then That interface to do notifications
#
# Note two: The implementation uses normally-closed reed switches
# from China. If you use normally open switches, you'll have to
# edit this code to invert the tests for the values coming from
# the sensors.
#
# Philip R. Moyer
# Adafruit
# May 2016
#
# This code is released under a BSD license and is in the public domain.
# Any redistribution must include the above header.
#####

#####
# Libraries
#####

import os
import string
import paho.mqtt.client as mqtt
import Adafruit_IO
import time
```

```
#####
# Globals
#####

# -- Change these as needed for your installatin --

localBroker = "YOUR_BROKER_IP" # Local MQTT broker
localPort = 1883 # Local MQTT port
localUser = "YOUR_MQTT_USERID" # Local MQTT user
localPass = "YOUR_MQTT_PASSWORD" # Local MQTT password
localTopic = "/security" # Local MQTT topic to monitor
localTimeOut = 120 # Local MQTT session timeout

adafruitUser = "YOUR_ADAFRUIT_IO_USERID" # Adafruit.IO user ID
adafruitKey = "YOUR_ADAFRUIT_IO_KEY" # Adafruit.IO user key
adafruitTopic = "alarms" # Adafruit.IO alarm topic

# -- You should not need to change anything below this line --

sensorList = {} # List of sensor objects

#####
# Classes and Methods
#####

class sensor():
    def __init__(self):
        self.name = "" # Name of sensor in MQTT
        self.humanName = "" # Human-meaningful name (e.g., "front door")
        self.lastSeen = 0 # Number of seconds since the sensor was last seen
        self.state = "unknown" # State of the object: unknown, open, or closed

    def setState(self, newstate):
        self.state = newState

    def getState(self):
        return self.state

    def resetHeartbeat(self):
        self.lastSeen = 0

    def setname(self, newName, humanName):
        self.name = newName
        self.humanName = humanName

    def getname(self):
        return self.humanName

    def checkState(self, newState):
        if ("unknown" == self.state):
            self.state = newState
            return 0
        else:
            if (newState != self.state):
                self.state = newState
                if ("closed" == self.state):
                    return -1
                else:
                    return 1
            return 0
```

```

class sensorList():
    def __init__(self):
        self.sensorList = {}

    def addSensor(self, sensorName, humanName):
        self.sensorList[sensorName] = sensor()
        self.sensorList[sensorName].setname(sensorName, humanName)

    def getSensorName(self, sensorID):
        return self.sensorList[sensorID].getname()

    def sensorState(self, sensorID, monitorState):
        rv = self.sensorList[sensorID].checkState(monitorState)
        if (0 != rv):
            # State changed!
            if (0 > rv):
                outBuf = "INFO "+self.getSensorName(sensorID)+" "+monitorState
                print(outBuf)
            else:
                outBuf = "ALARM "+self.getSensorName(sensorID)+" "+monitorState
                print(outBuf)
            print("Initiating connection to Adafruit.IO")
            AIOclient = Adafruit_IO.MQTTClient(adafruitUser, adafruitKey)
            print("Setting callbacks for Adafruit.IO")
            AIOclient.on_connect = AIOconnected
            AIOclient.on_disconnect = AIOdisconnected
            AIOclient.on_message = AIOmessage
            print("Connecting to Adafruit.IO")
            AIOclient.connect()
            time.sleep(5)
            print("Publishing outBuf")
            # AIOclient.publish("alarms", outBuf)
            AIOclient.publish("alarms", outBuf)
            print("Disconnecting")
            AIOclient.disconnect()

```

```
#####
```

```
# Functions
```

```
#####
```

```
# Callback functions for Adafruit.IO connections
```

```

def AIOconnected(client):
    # client.subscribe('alarms')
    print("Connected to Adafruit.IO")

```

```

def AIOdisconnected(client):
    print("adafruit.io client disconnected!")

```

```

def AIOmessage(client, feed_id, payload):
    print("adafruit.io received ", payload)

```

```

# returnState takes a numeric voltage value from the sensor and
# returns the state of the monitored device. With a voltage divider
# that uses a 1M ohm R1 and a 470K ohm R2, the "closed" state returns
# 1024 and the open state returns between 1 and 40.

```

```

def returnState(inVal):
    if (1000 < inVal):
        return "closed"
    if (100 > inVal):

```

```

return "open"
else:
    return "unknown"

#####
# Main
#####

if "__main__" == __name__:
    # Set timer

    sensList = sensorList()
    sensList.addSensor("security_001", "front door")

    # The callback for when the client receives a CONNACK response from the server.
    def on_connect(client, userdata, flags, rc):
        print("Connected with result code "+str(rc))

    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.
    client.subscribe("/security")

    # The callback for when a PUBLISH message is received from the server.
    def on_message(client, userdata, msg):
        (sensorID, sensorVoltage) = string.split(msg.payload)
        sensorVoltage = string.atoi(sensorVoltage)
        sensorName = sensList.getSensorName(sensorID)
        sensList.sensorState(sensorID, returnState(sensorVoltage))
        # print(sensorName+" "+returnState(sensorVoltage))

    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_message = on_message

    client.connect(localBroker, localPort, localTimeout)

    # Blocking call that processes network traffic, dispatches callbacks and
    # handles reconnecting.
    # Other loop*() functions are available that give a threaded interface and a
    # manual interface.
    client.loop_forever()

    quit()

```

Copy *security.py* to */home/pi/security.py* on the machine that is your MQTT broker. Edit the program so the parameters are correct for your installation. I recommend keeping the */security* topic, though.

You also need to install the Adafruit\_IO Python library. You can clone it from the [GitHub repository](https://github.com/adafruit/Adafruit_IO_Python) (<http://adafru.it/osB>) or click this button to download a Zip file. Put it in the same directory as your *security.py* program.

[Download a Zip file of the Adafruit\\_IO Python library](http://adafru.it/osC)  
<http://adafru.it/osC>

## Step 4 - Configure security.py to Run at Boot

I cheat and use the old school mechanism for starting *security.py* at boot: I add it to */etc/rc.local* and make sure

*rc.local* runs at boot time. This must be done as root. Here are the commands to use to make this configuration:

```
sudo chmod 755 /etc/rc.local  
sudo nano /etc/rc.local
```

You can use your favorite text editor to make changes to */etc/rc.local*. All you need to do is add one line near the bottom of the file. Here is the bottom of my *rc.local* file on my Raspberry Pi that controls the alarm system:

```
ifup eth0
```

```
/usr/bin/python /home/pi/security.py &
```

```
exit 0
```

The line with python that executes *security.py* is the line to add.

## Step 5 - Start security.py

This is the easiest part. Just run this command:

```
sudo /etc/rc.local
```

Next: [Configuring io.adafruit.com](https://learn.adafruit.com/diy-esp8266-home-security-with-lua-and-mqtt)



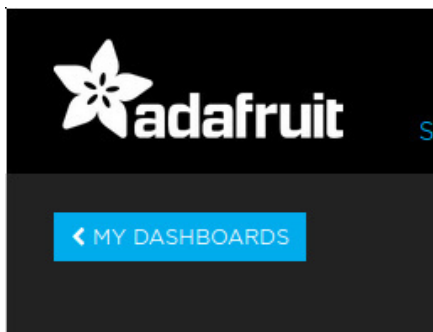
# Configuring io.adafruit.com

In this section, we'll walk through the steps necessary to configure io.adafruit.com to receive your alarm data. Fortunately, it's pretty straightforward and works *extremely* well.

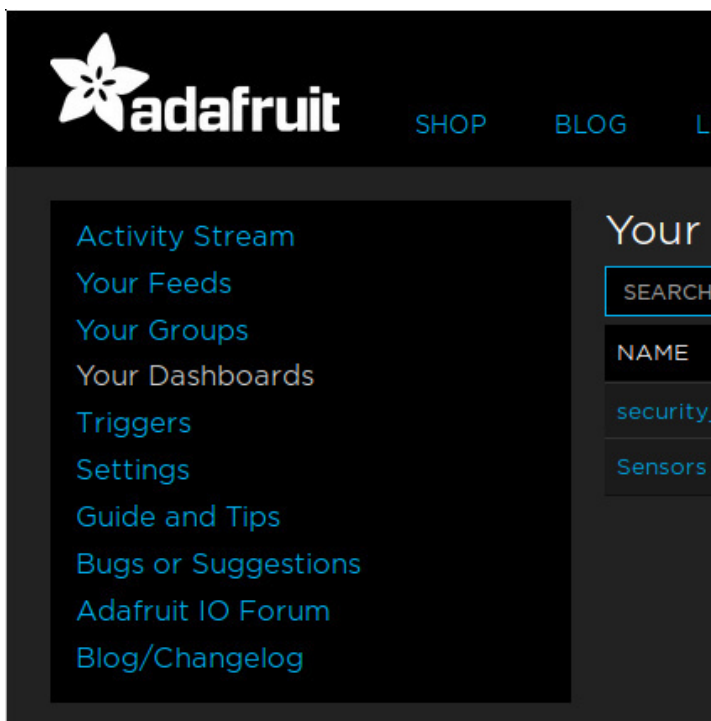
## Step 1 - Create a feed for your alarm data

Go to io.adafruit.com and log in with your credentials. If you don't have an account there yet, you can request one from that page. Your io.adafruit.com credentials will be the same as they are for other parts of the Adafruit environment.

Once you have logged in, you'll need to create the feed. Navigate away from your default landing page by clicking the "My Dashboards" button:

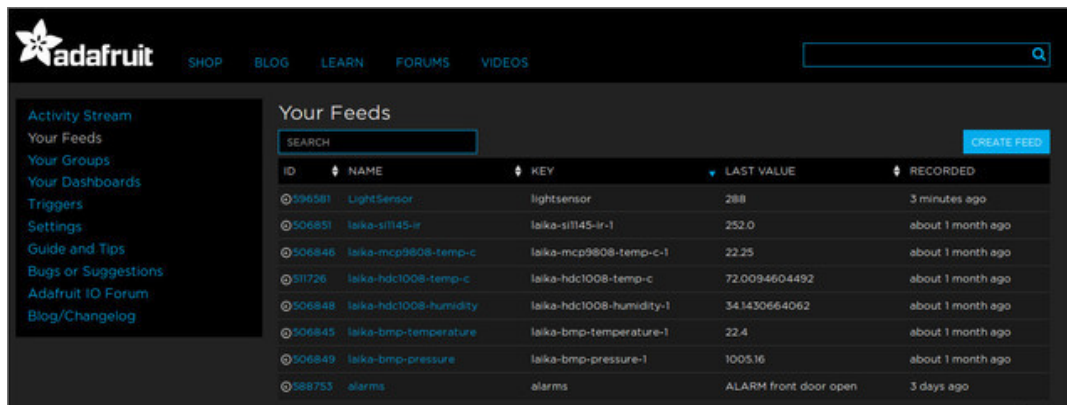


This will bring you to a screen that has a site navigation bar on the left side. It will look like this:



Click on the "Your Feeds" link (second from the top) to take you to your feeds management page. Once there,

you'll see a page that looks like this, except it will have your feeds instead of mine:



Once you're at this point, create a new feed by clicking the light blue button on the far right, labeled "Create Feed." If you look closely at the image above, you'll see that I already have a feed titled "alarms." This is the feed you need to create first. The feed creation page should look like this for you:

**CREATE A NEW FEED**

**NAME**

alarms

**DESCRIPTION**

Security system alarm feeds.

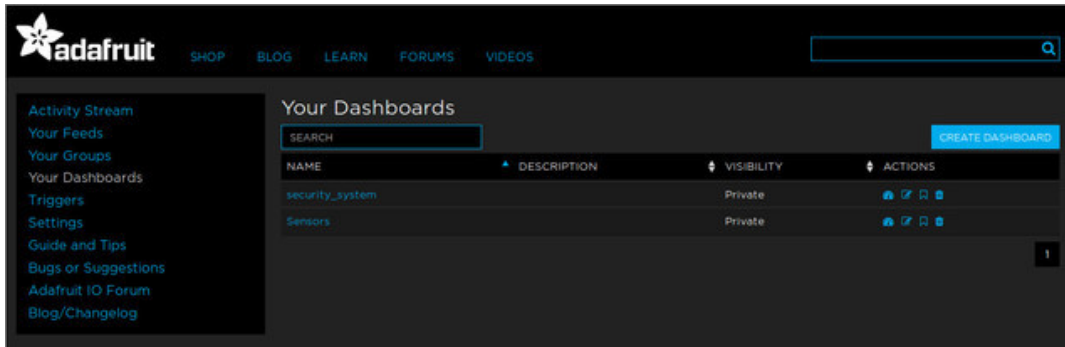
**CANCEL** **CREATE FEED**

When you have entered the feed information as shown (I recommend you keep the "alarms" feed name and will assume you have done so) click the "Create Feed" button in the lower right to create your feed. The io.adafruit.com system has a feed limit in place right now, so you may need to delete another feed (or two) to create the new alarms feed. I am hopeful that with the success of Adafruit.IO we'll soon have a paid tier available where we can buy extra feeds and dashboards (but that's not up to me - I'm just a wishful customer).

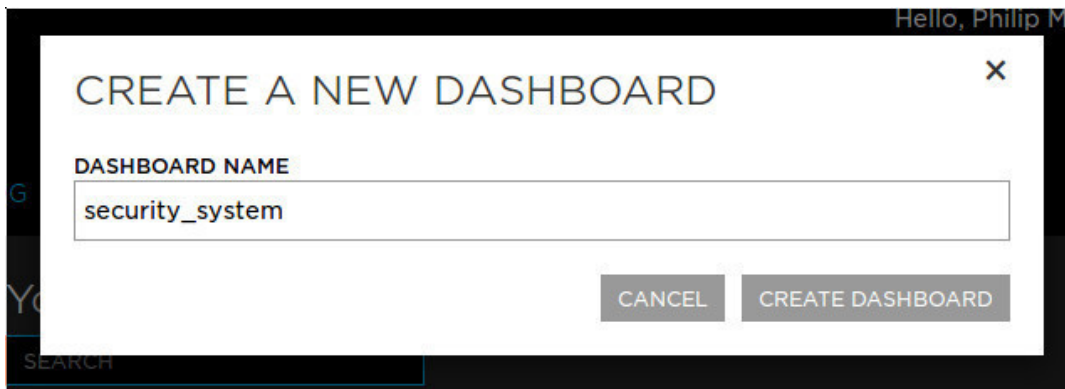
## Step 2 - Create the dashboard for the alarms

The next step is to create the dashboard to display your feed information. Remember, the "smarts" of the system are in the *security.py* Python program. We don't need to create separate feeds for every alarm sensor we have, as we would need to do for some other projects. The Python program will essentially "multiplex" all the alarms into this single feed.

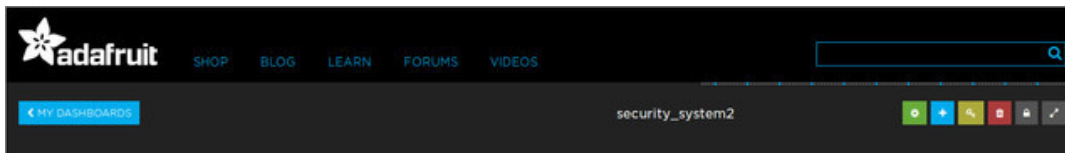
Back in the navigation section on the left, click the button labeled "Your Dashboards." It's the fourth one down in the list. This will bring you to the dashboard management page, which looks like this:



Click on the light blue "Create Dashboard" button on the far right. This will bring up the new dashboard creation page. It should look similar to this:



Enter the name for your new dashboard (I used "security\_system") and click the "Create Dashboard" button in the lower right. You'll be shown a blank dashboard page like this:



(You can see that I used the name "security\_system2" since I already had a dashboard for "security\_system.") See the row of colored boxes/buttons on the right? The blue one, second from the left, will create a new block object on the dashboard. Click it. You'll then see a dialog box with all the different kinds of blocks you can add to your dashboard. The one we want is the text block. It's the fifth one down from the top:

## CREATE A NEW BLOCK



A toggle button is useful if you have an ON or OFF type of state. You can configure what values are sent.

CREATE



A momentary button works similarly to a hardware push button.

CREATE



The slider works well if you have a range of values you need to send.

CREATE



A gauge is a read only block type that shows a fixed range of values.

CREATE



A text block can be used to send data as well as view data.

CREATE



A stream block can be used to view the rolling history of data for multiple feeds.

CREATE



An image block can be used to view base64 encoded images.

CREATE



The line chart is used to chart one or more feeds.

CREATE



The color picker is used to send or view color values in hex format.

Click the "Create" button next to the text block. This will bring up another dialog box that lets you select which feeds will appear in the block. We only want one feed - the "alarms" feed you created above.

CREATE A NEW BLOCK

STEP 1: CHOOSE BLOCK TYPE

STEP 2: CHOOSE FEEDS

Add up to 1 feed

SEARCH

NEW FEED NAME

CREATE

FEED/GROUP

LAST VALUE

RECORDED

ACTION

My Feeds

laika-bmp-tempe...	22.4	about 1 month ago	CHOOSE
laika-mcp9808-te...	22.25	about 1 month ago	CHOOSE
laika-hdc1008-hu...	34.1430664062	about 1 month ago	CHOOSE
laika-bmp-pressu...	1005.16	about 1 month ago	CHOOSE
laika-si1145-ir	252.0	about 1 month ago	CHOOSE
laika-hdc1008-te...	72.0094604492	about 1 month ago	CHOOSE
alarms	ALARM front door ...	3 days ago	CHOOSE
LightSensor	291	less than 5 seconds...	CHOOSE

NEXT STEP >

Click the "Choose" link next to your "alarms" feed. Then click the blue "Next Step" button in the lower right corner.

© Adafruit Industries

<https://learn.adafruit.com/diy-esp8266-home-security-with-lua-and-mqtt>

Page 25 of 38

CREATE A NEW BLOCK

STEP 1: CHOOSE BLOCK TYPE EDIT

STEP 2: CHOOSE FEEDS EDIT

STEP 3: BLOCK SETTINGS

BLOCK TITLE

alarms

FONT SIZE

MEDIUM ▾

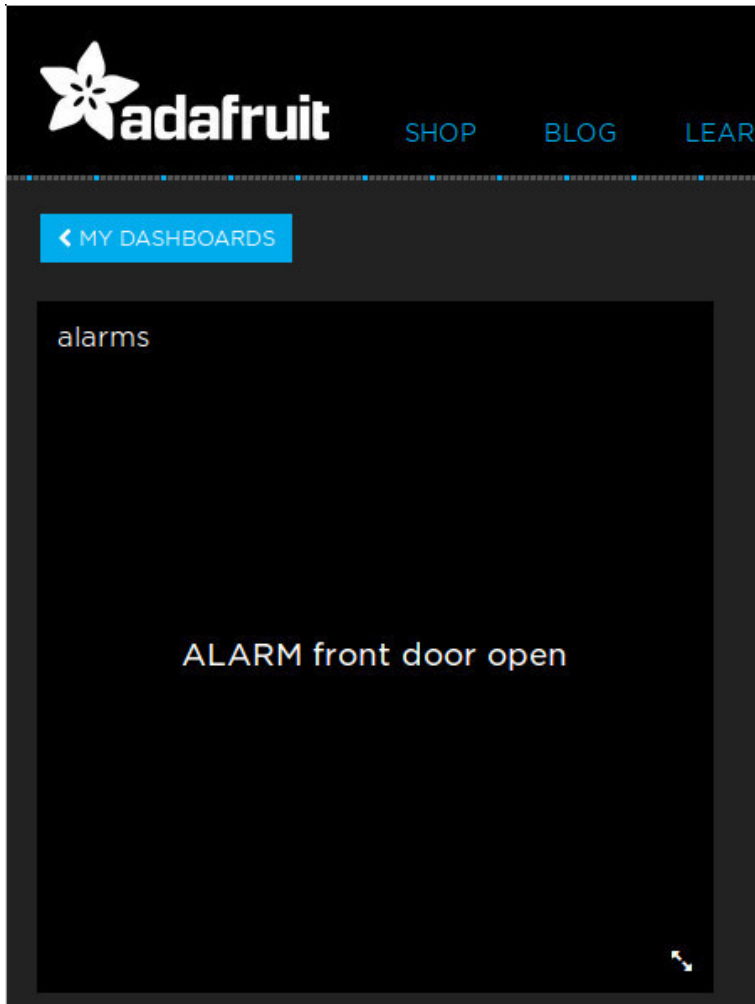
BLOCK PREVIEW

alarms

CANCEL CREATE BLOCK

This brings you to the final configuration dialog for your new block. You can change the font size or name that's displayed on the box. I just left the defaults, but you can put what you like here. Whatever is most aesthetically pleasing to you. When you are done editing the information, click the light blue "Create Block" button in the lower right.





The default text box size is a little small, so grab the double arrow in the bottom right corner and resize the text block until it looks right to you (you can always tune it up later if it's not quite right.)

That's it! Your security system alarms should now appear in this text block on your dashboard! You can test it by opening a door or window and then checking to see if the alarm shows up.

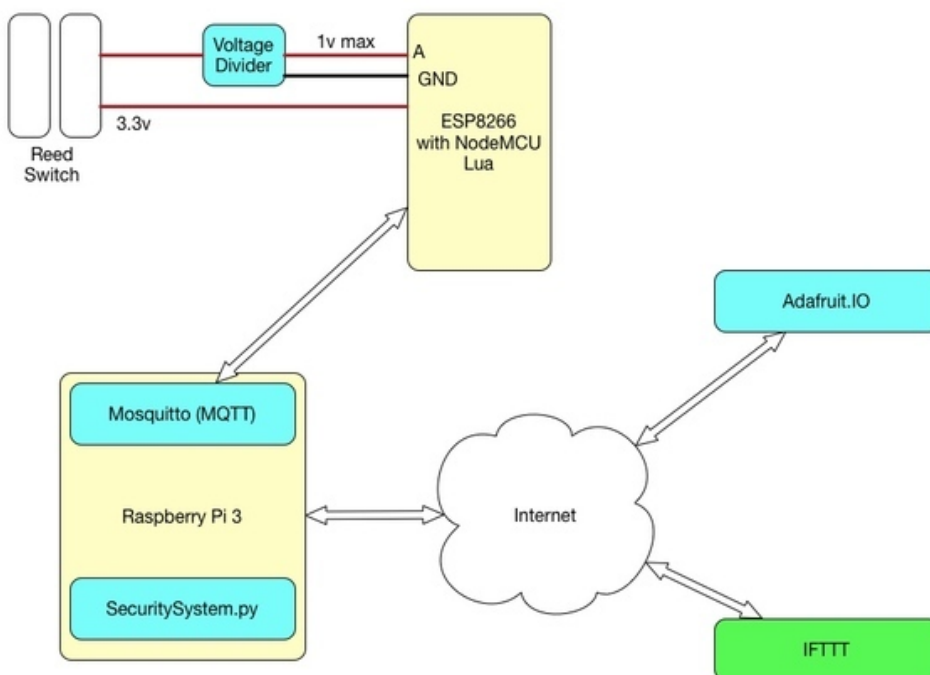
## Troubleshooting

If you don't receive alarms, there are a few things you can check.

1. Does the ESP8266 have power? (Don't judge.)
2. Is the ESP9266 properly connecting to your network? You can check this by connecting a micro-USB cable to the device and opening it with CoolTerm or another terminal program.
3. Is Mosquitto running on your Raspberry Pi, and does it have the proper topic names? You can use the `mosquitto_sub` command to check this.
4. Is your `security.py` Python program configured correctly and communicating with `io.adafruit.com`? Remember, you'll need to make sure your Adafruit.IO key is copied correctly into the program.
5. Finally, are all your names properly matched up (e.g., "alarm" in one place but "alarms" in another)?

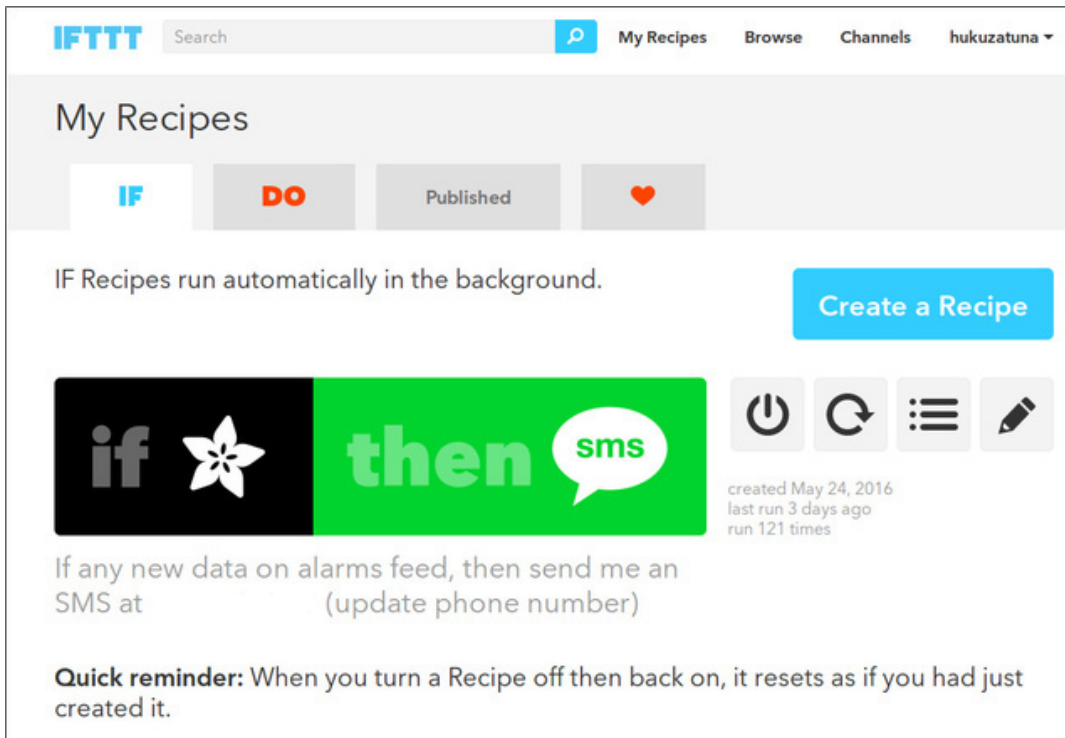
Next: Configuring IFTTT to text

# Configuring IFTTT to Text



Remember this diagram from the start of the tutorial? The green box in the lower right is If This Then That ([ifttt.com](http://ifttt.com) (<http://adafru.it/fYY>)), a web site that monitors streams for events and takes actions based on them. If This Then That (IFTTT) uses programs called *recipes* to react to things on the Internet. There are two components to the recipe we'll use that are important: one is the ability for IFTTT to read an [io.adafruit.com](http://io.adafruit.com) topic, and the other is the ability to send SMS text messages (note that IFTTT imposes a 100 message per month limit).

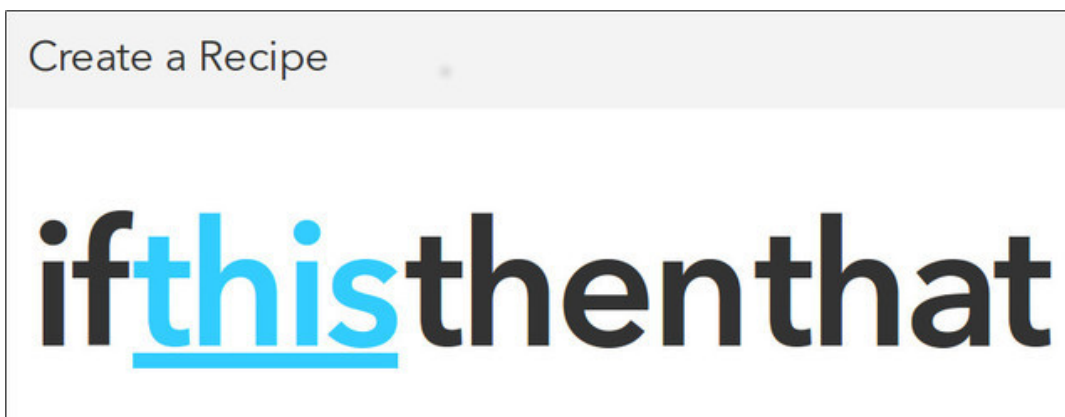
Here is what your page will look like when you're done (except it will show your phone number):



Let's get started making the recipe!

## Step 1 - Add the Adafruit.IO monitor

Click the light blue "Create a Recipe" button on the right side of the landing page. You can see it in the above image. This will bring you to a page that looks like this:



The first step is to define the trigger that will cause the recipe to run. To do that, click the "this" link.

## Choose Trigger Channel step 1 of 7

Showing Channels that provide at least one Trigger. [View all Channels](#)



500px



abode



Adafruit



Amazon Alexa



Android Battery



Android Device



Android Location



Android Phone Call



Android Photos



Android SMS



Android Wear



AppZapp



ARTIK Cloud



Now, I already have Adafruit as one of my trigger channels because I've already built the alarm system. You will probably need to use the search function to look for "Adafruit," or click the "view all channels" link. Find Adafruit and add it to your list of trigger channels. Then, click it to select it.

This will take you to the second recipe configuration step:



## Choose a Trigger

step 2 of 7

### Any new data

This Trigger fires any time there is new data in your feed.

### Monitor a feed on Adafruit IO

This Trigger fires anytime it validates the data that you send to your feed.  
Example: If Feed Temperature > 80, fire Trigger.

You want to click on "Monitor a feed on Adafruit IO" (the one on the right) for your security system. The next dialog will look something like this:



## Complete Trigger Fields

step 3 of 7

Monitor a feed on Adafruit IO



Feed

alarms



Relationship

not equal to



Value

INFO

The value to compare against.

Create Trigger

In the dropdown menus, select your alarm feed, "not equal to," and enter "INFO" in the text box. This will ensure that any text that says "ALARM" will get sent to you. When you have this dialog box configured correctly, click the "Create Trigger" button.



There! Now you've successfully created the trigger for your alarm notification!

Next, we'll configure the response to the trigger. Click the blue "that" link....

## Choose Action Channel

step 4 of 7

back

Showing Channels that provide at least one Action. [View all Channels](#)

Search Channels



500px



abode



Adafruit



Amazon Cloud Drive



Android Device



Android SMS



Android Wear



Arlo



ARTIK Cloud



Assistant.ai



AT&T M2X



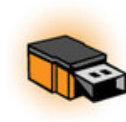
Bang & Olufsen's BeoLink



Beeminder



bitly



blink(1)



Blogger



BMW Labs



Box



The action channel you want is labeled "SMS." It didn't appear in my default list, so I had to search for it just like I had to search for the Adafruit trigger. When you find SMS, add it to your list of action channels and select it. This will open a dialog page with only one option to select - "Send me an SMS."



## Choose an Action

step 5 of 7

### Send me an SMS

This Action will send an SMS to your mobile phone.

Select it. That will bring up the next step, which is configuring the message you want to send when you get an alarm. I use the default message, though you're obviously free to experiment with whatever format you like.





## Complete Action Fields

step 6 of 7

Send me an SMS



Message

FeedName FeedValue is Operator  
TriggerValue !

Create Action

When you're happy with the formatting, click the "Create Action" button. This will bring up the final page in the IFTTT configuration. I already have my phone number configured in IFTTT, so I can't easily show you what the last page looks like. You'll need to enter your mobile number and verify it, after which you'll be all set!

That's it! Now you've written an IFTTT recipe to monitor your io.adafruit.com feed and text you a message when it sees an alarm!

Next: Alternative designs and enhancements

# Alternative Designs and Enhancements

There are a number of weaknesses in the security system. I'll list them here, along with the enhancements that will address those weaknesses. Then I'll briefly talk about some alternative architectures that you can build instead of the one I described in this tutorial. First, the weaknesses and fixes....

- **It won't work if the power is cut.** Maybe the intruders are smart and cut the power to the building/apartment/office before they break in. Solution: LiPo batteries on each Feather ESP8266 so the sensors will continue working even without USB power. Also, a big 12v battery with an inverter and 5v output connectors to run the wireless access point and MQTT broker server even without mains power (I use a DuraCell AGM 1300 Amp PowerPack Pro).
- **It won't work if the Internet connection is cut.** The intruders might be unusually clever and cut both the power and the Internet. Solution: the Raspberry Pi and WiFi will be running from the PowerPack Pro, but the alarms need to get to Adafruit.IO. The solution to that is to fall back to a FONA 800L breakout board. To do this, we'd have to modify the Python program to ping [adafruit.com](http://adafruit.com) prior to making the connection. If the ping doesn't work, the program can fire up the FONA board and connect to Adafruit.IO that way.
- **Adafruit.IO may be down.** Solution: use a backup cloud-based message queue, such as Amazon Web Services (AWS) SQS. This will allow us to test for a successful topic publication and, if it failed, connect to AWS and publish to an alternative queue. This would also require some added expense (though not much) and additional complexity in the program and IFTTT recipes.
- **The intruders might put a very strong magnet outside the door/window close to the switch so they can open the door or window without triggering the alarm.** Solution: put Raspberry Pi cameras near the points of ingress and program them to detect motion. I haven't actually tried this myself, but I've seen tutorials online that describe how to do this. It would be fairly straightforward to program the additional Raspberry Pis to send alarms to Adafruit.IO if they detect motion, but the door/window sensors haven't been triggered. This would also allow us to take pictures of the intruders that we can give to the police.
- **Security systems are supposed to have cameras** Solution: add some cameras, as noted above.
- **It would be good to have smoke and carbon monoxide detectors.** I agree! Solution: use MQ- series gas sensors to detect smoke and carbon monoxide. Power them from an independent 5v supply and monitor them with a Raspberry Pi. This will require connecting the MQ sensors through an [ADS1115 16-bit ADC \(http://adafru.it/1085\)](http://adafruit.com/product/1085) (Adafruit Product ID 1085) to convert the output voltage from the analog MQ sensors to a digital signal the Raspberry Pi can read.

Finally, here are some alternative design ideas.

Instead of using a Feather ESP8266 for each sensor like I designed, you can save a little money by hardwiring 16 sensors at a time through a signal multiplexer board (these are available from various providers online). This will allow you to run wired connections to each door/window switch and read them directly from the Raspberry Pi via an ADC. I personally wouldn't do this because I hate running long strands of wire all over.

Skip the Raspberry Pi running the MQTT broker and security program. You can do this by doing more clever Lua programming than I've done and keeping track of the sensor state in the ESP8266 itself. Then it can connect directly to Adafruit.IO (I think - haven't tried this myself) to send an alarm when necessary. The IFTTT process would be exactly the same. I didn't do it this way because I'm only now teaching myself Lua and I don't think my Lua skills are up to the task yet.

Let me know if you have other ideas!

Next: the appendix - how to re-flash your ESP8266



# How to Re-flash Your ESP8266

As you work with and learn Lua on the ESP8266, it is likely, because of some quirks in the language and processor implementatoin, that you'll eventually create a tight loop in your *init.lua* file. That, or an infinite loop. Either one will brick your ESP8266. Hint: if you need to do something repeatedly and quickly (like checking the sensor state on a door/window open sensor), use the built-in timer function to set an interrupt with a callout function! Failing to understand that constraint is how I bricked my first ESP8266 and consequently learned how to re-flash the device.

## Step 1 - Getting the firmware

Now, you *could* go and build the complete toolchain necessary to cross-compile the firmware for the ESP8266. I might do that sometime just for the experience. Also, you *could* go to the [nodemcu.com](http://adafru.it/f1G) (<http://adafru.it/f1G>) web site and try to download the current firmware. I, however, found the organization of the FTP site confusing and I couldn't find the right firmware.

Don't despair, though! A fantastic individual named Marcel Stor has created an interactive web page that allows you to [build your own custom ESP8266 firmware](http://adafru.it/osD) (<http://adafru.it/osD>)! Just click that link and it will take you to the firmware configuration page.

Enter your e-mail address so you can receive notificatin when your bild is done and ready for download. Then scroll down to the configuration section. This is what it looks like:

### Select branch to build from

☒ master <> ☐ dev <>

Click the <> to verify on GitHub that the selected branch actually contains what you expect it to.

Note that only active branches are displayed here. I don't want you to work on branches the NodeMCU team doesn't support anymore.

### Select modules to include

- |  |   |   |   |
|--|---|---|---|
| <input type="checkbox"/> ADC <a href="#">[?]</a>     | <input type="checkbox"/> end user setup <a href="#">[?]</a>   | <input type="checkbox"/> 1-Wire <a href="#">[?]</a>       | <input type="checkbox"/> SPI <a href="#">[?]</a>              |
| <input type="checkbox"/> AM2320 <a href="#">[?]</a>  | <input checked="" type="checkbox"/> file <a href="#">[?]</a>  | <input type="checkbox"/> perf <a href="#">[?]</a>         | <input type="checkbox"/> struct <a href="#">[?]</a>           |
| <input type="checkbox"/> APA102 <a href="#">[?]</a>  | <input checked="" type="checkbox"/> GPIO <a href="#">[?]</a>  | <input type="checkbox"/> PWM <a href="#">[?]</a>          | <input checked="" type="checkbox"/> timer <a href="#">[?]</a> |
| <input type="checkbox"/> bit <a href="#">[?]</a>     | <input type="checkbox"/> HTTP <a href="#">[?]</a>             | <input type="checkbox"/> RC (no docs) <a href="#">[?]</a> | <input type="checkbox"/> TSL2561 <a href="#">[?]</a>          |
| <input type="checkbox"/> BME280 <a href="#">[?]</a>  | <input type="checkbox"/> HX711 <a href="#">[?]</a>            | <input type="checkbox"/> rotary <a href="#">[?]</a>       | <input type="checkbox"/> U8G <a href="#">[?]</a>              |
| <input type="checkbox"/> BMP085 <a href="#">[?]</a>  | <input type="checkbox"/> I <sup>2</sup> C <a href="#">[?]</a> | <input type="checkbox"/> RTC fifo <a href="#">[?]</a>     | <input checked="" type="checkbox"/> UART <a href="#">[?]</a>  |
| <input type="checkbox"/> CJSON <a href="#">[?]</a>   | <input type="checkbox"/> mDNS <a href="#">[?]</a>             | <input type="checkbox"/> RTC mem <a href="#">[?]</a>      | <input type="checkbox"/> UCG <a href="#">[?]</a>              |
| <input type="checkbox"/> CoAP <a href="#">[?]</a>    | <input type="checkbox"/> MQTT <a href="#">[?]</a>             | <input type="checkbox"/> RTC time <a href="#">[?]</a>     | <input checked="" type="checkbox"/> WiFi <a href="#">[?]</a>  |
| <input type="checkbox"/> crypto <a href="#">[?]</a>  | <input checked="" type="checkbox"/> net <a href="#">[?]</a>   | <input type="checkbox"/> Sigma-delta <a href="#">[?]</a>  | <input type="checkbox"/> WS2801 <a href="#">[?]</a>           |
| <input type="checkbox"/> DHT <a href="#">[?]</a>     | <input checked="" type="checkbox"/> node <a href="#">[?]</a>  | <input type="checkbox"/> SNTP <a href="#">[?]</a>         | <input type="checkbox"/> WS2812 <a href="#">[?]</a>           |
| <input type="checkbox"/> encoder <a href="#">[?]</a> |   |   |   |

Click the [\[?\]](#) to go to the module documentation if you're uncertain whether you should include it or not.

The selected default modules will give you a basic firmware to start with. Select as few modules as possible as to keep the firmware small. See the [FAQ](#).

I'd really like to offer some guidance as to which modules to select but the NodeMCU team doesn't provide a documented dependency matrix yet. See [#386](#) for details.

### Miscellaneous options

- ☐ SSL support, TLS 1.1 with [4 cipher suites](#)
- ☐ debug ON, *turn it on only if you're troubleshooting an issue with NodeMCU developers as it will increase the size of the firmware and decrease its performance*

Build from the Master branch unless you *absolutely* know you need dev for some reason. Also, I leave the Miscellaneous Options unchecked. In addition to the defaults, I think you'll want to add:

- ADC
- Bit
- MQTT
- Perf
- PWM

As you play more with the ESP8266 and Lua, and become ever more enamored of the platform, you'll probably want to return to this page and build specific firmware for sensors you'd like to deploy outside of the security system project.

Once you have the options configured to your satisfaction, click the blue "Start your build" link at the bottom of the page. The site will send you e-mail when your build starts, and again when it's ready to download.

There will be two firmware files: a hardware float version and an integer version. I download both, but I only use the float version.

Next you'll need to flash the firmware onto your device.

## Step 2 - Getting the tool to flash an ESP8266

As I mentioned, I'm on a Mac, and the most convenient tool for me to use is called *desptool*. This is a Python program (so it should run on any platform that supports Python) that you can clone from its [GitHub repository](https://github.com/adafruit/desptool) (<http://adafru.it/osE>). I put my firmware builds in the same directory *asesptool.py*, just for convenience.

## Step 3 - Flash the ESP8266

If you are using a Feather M0 ESP8266 or Huzzah ESP8266 Breakout, you'll need to wire Pin 0 to Ground. This puts the device in bootloader mode. I use a female-female DuPont wire for this. Some boards from other vendors do not require this (in fact, many of the boards I've found arrive without any firmware at all, so the first thing you have to do is flash them.)

Once the board is ready to flash, you need to run *desptool.py* with the correct arguments. On the Mac, the correct command is

```
python ./esptool.py --port /dev/tty.SLAB_USBtoUART write_flash 0x000000 <path to firmware>
```

Of course, replace *<path to firmware>* with your actual firmware path. It takes less than a minute to load the new firmware into flash memory on the ESP8266. Once you're done with that, you'll have a working board (or, a working board *again*) and you can get on with your project!